ROHDE&SCHWARZ

Measuring Instruments
and Systems Division

Manual

R&S BASIC Interpreter

Version 2.2

Printed in the Federal
Republic of Germany

376.1546.32 - 02 -

Certified Quality System

ISO 9001

DQS REG. NO 1954-04

Qualitätszertifikat

Sehr geehrter Kunde,

Sie haben sich für den Kauf eines Rohde & Schwarz-Produktes entschieden. Hiermit erhalten Sie ein nach modernsten Fertigungsmethoden hergestelltes Produkt. Es wurde nach den Regeln unseres Qualitätsmanagementsystems entwickelt, gefertigt und geprüft. Das Rohde & Schwarz-Qualitätsmanagementsystem ist nach ISO 9001 zertifiziert.

Certificate of quality

Dear Customer,

You have decided to buy a Rohde & Schwarz product. You are thus assured of receiving a product that is manufactured using the most modern methods available. This product was developed, manufactured and tested in compliance with our quality management system standards.
The Rohde & Schwarz quality management system is certified according to ISO 9001.

Certificat de qualité

Cher client,

Vous avez choisi d'acheter un produit Rohde & Schwarz. Vous disposez donc d'un produit fabriqué d'après les méthodes les plus avancées. Le développement, la fabrication et les tests respectent nos normes de gestion qualité.
Le système de gestion qualité de Rohde & Schwarz a été homologué conformément à la norme ISO 9001.

095.1000 Bl.0-0796

ROHDE&SCHWARZ

ROHDE&SCHWARZ

ROHDE & SCHWARZ GmbH & Co. KG
Postfachadresse: Postfach 80 14 69 · D-81614 München
Hausadresse: Mühldorfstraße 15 · D-81671 München
Telefon: (München 089) 41 29-0 · International: (4989) 41 29-0
Telefax: (München 089) 41 29-21 64

ROHDE & SCHWARZ GmbH & Co. KG
Werk Köln
Graf-Zeppelin-Straße 18
D-51147 Köln
Tel. (0 22 03) 49-0
Telefax (0 22 03) 49-229

ROHDE & SCHWARZ GmbH & Co. KG
Werk Teisnach
Kaikenrieder Straße 27
D-94244 Teisnach
Tel. (0 99 23) 28-0
Telefax (0 99 23) 28-174

ROHDE & SCHWARZ MESSGERÄTEBAU GmbH
Riedbachstraße 58
D-87700 Memmingen
Tel. (0 83 31) 108-0
Telefax (0 83 31) 108-124

ROHDE & SCHWARZ Engineering and Sales GmbH
Mühldorfstraße 15
D-81671 München
Tel. (0 89) 41 29-37 11
Telefax (0 89) 41 29-37 23

R&S BICK Mobilfunk GmbH
Im Landerfeld 7
D-31848 Bad Münder
Tel. (0 50 42) 998-0
Telefax (0 50 42) 998-105

ROHDE & SCHWARZ FTK GmbH
Wendenschloßstraße 168, Haus 28
D-12557 Berlin
Tel. (0 30) 6 58 91-0
Telefax (0 30) 65 55 02 21

SIT Gesellschaft für Systeme
der Informationstechnik mbH
Wendenschloßstraße 168, Haus 28
D-12557 Berlin
Tel. (0 30) 6 58 91-222
Telefax (0 30) 6 58 80-183

Zweigniederlassungen — Subsidiaries in Germany

Zweigniederlassung Berlin
Ernst-Reuter-Platz 10
D-10587 Berlin
Tel. (0 30) 34 79 48-0
Telefax (0 30) 34 79 48-48

Zweigniederlassung Büro Bonn
Josef-Wirmer-Straße 1–3
D-53123 Bonn
Tel. (02 28) 9 18 90-0
Telefax (02 28) 25 50 87

Zweigniederlassung Dresden
Fritz-Reuter-Straße 32c
D-01097 Dresden
Tel. (03 51) 4 45 92-0
Telefax (03 51) 4 45 92-15

Zweigniederlassung Hamburg
Steilshooper Allee 47
D-22309 Hamburg
Tel. (0 40) 63 29 00-0
Telefax (0 40) 6 30 78 70

Zweigniederlassung Karlsruhe
Am Sandfeld 9
D-76149 Karlsruhe
Tel. (07 21) 9 78 21-0
Telefax (07 21) 9 78 21-41

Zweigniederlassung Köln
Graf-Zeppelin-Straße 18
D-51147 Köln
Tel. (0 22 03) 8 07-0
Telefax (0 22 03) 8 07-50

Zweigniederlassung München
Mühldorfstraße 15
D-81671 München
Tel. (0 89) 41 86 95-0
Telefax (0 89) 40 47 64

Zweigniederlassung Neu-Isenburg
Siemensstraße 20
D-63263 Neu-Isenburg
Tel. (0 61 02) 20 07-0
Telefax (0 61 02) 80 00 40

Zweigniederlassung Telekommunikation
Siemensstraße 20
D-63263 Neu-Isenburg
Tel. (0 61 02) 20 07-0
Telefax (0 61 02) 20 07-12

Zweigniederlassung Nürnberg
Donaustraße 36
D-90451 Nürnberg
Tel. (09 11) 6 42 03-0
Telefax (09 11) 6 42 03-33

R&S International

		Telephone Telefax E-mail
Argentina	Precisión Electrónica SRL Esmeralda 582, Piso 4, Off. 11 **1007 Buenos Aires**	(1) 3944815 (1) 3272332
Australia	ROHDE & SCHWARZ (Australia) Pty. Ltd. 63 Parramatta Road **Silverwater, N.S.W. 2141**	(2) 97480155 (2) 97481836
Austria	ROHDE&SCHWARZ ÖSTERREICH Ges.m.b.H. Sonnleithnergasse 20 **A-1100 Wien**	(1) 6026141 (1) 6026141-14
Azerbaijan	ROHDE&SCHWARZ Representative Office Azerbaijan avenue 35 **370139 Baku**	(412) 933138 (412) 930314
Bangladesh	Business International Ltd. 146/A, New Bailey Rd., P.O.B. 727 **Dhaka-2**	(2) 839046 (2) 833520
Belgium	ROHDE & SCHWARZ BELGIUM N.V. Excelsiorlaan 31 Bus 1 **B-1930 Zaventem** isabel.Veiro/RSB%RSB@rsd.de	(2) 7215002 (2) 7250936
Brazil	ROHDE & SCHWARZ Precisão Eletrônica Ltda. Rua Geraldo Flausino Gomes, 42-1° and. **04575-060 São Paulo – SP**	(11) 55052177 (11) 55055793
Brunei	GKL Equipment Pte. Ltd. # 11-01 BP Tower, 396, Alexandra Rd. **Singapore 0511**	2760626 2760629
Bulgaria	ROHDE & SCHWARZ Representation Office 39, Fridtjof Nansen Blvd. **1000 Sofia**	(2) 655133 (2) 656833
Canada	COM: ROHDE & SCHWARZ CANADA INC. 555 March Rd. **Kanata, Ontario K2K 2M5**	 (613) 5928000 (613) 5928009
	T & M: TEKTRONIX CANADA INC. 785 Arrow Road **Weston, Ontario M9M2L4**	 (416) 747-5000 (416) 747-7581
Chile	DYMEQ Ltda. Avenida Larrain 6666 **Santiago**	(2) 2775050 (2) 2278775
China	ROHDE & SCHWARZ Repr. Office Room 821 Beijing Towercrest Plaza No. 3 Mai Zi Dian West Road Chao Yang District **Beijing 100016**	(10) 64672365 (10) 64672315
Czech Republic	ROHDE & SCHWARZ Praha, s.r.o. Pod kastany 3 **CZ-16000 Praha 6**	(2) 24322014 (2) 24317043
Denmark	ROHDE & SCHWARZ DANMARK A/S Ejby Industrivej 40 **DK-2600 Glostrup**	(43) 436699 (43) 437744
Ecuador	Digitec Ltd. El Heraldo 121 y El Dia **Quito**	(2) 430373 (2) 443782
Finland	Orbis Oy Vanha Kaarelantie 9 **FIN-01610 Vantaa**	(9) 478830 (9) 531604 info@orbis.fi
France	ROHDE & SCHWARZ FRANCE 25–27, rue J. Braconnier **F-92366 Meudon-La-Forêt Cédex**	(0) 141361000 (0) 141361010
	ROHDE & SCHWARZ FRANCE Agences régionales:	
	Rennes Sigma 1 rue du Bignon **F-35135 Chantepie**	(0) 299519700 (0) 299419131
	Toulouse Technoparc 3 B.P. 501 **F-31674 Labège Cédex**	(0) 561391069 (0) 561399910

1 Operating BASIC

1.1 The R&S BASIC Interpreter

The R&S-BASIC is an interpreter, i.e. the program produced by the user with the editor can already be executed without further compilation or assembling. This has the advantage that written programs can be immediately tested and edited if required.

BASIC is a computer language used worldwide. However, BASICs from different computer manufacturers differ with respect to the instruction syntax and range which means that BASIC programs must first be rewritten or adapted before they can be used on another computer.

The R&S BASIC used is based on the standard or on widely used BASICs similar to the standard. The core of BASIC as defined in the ANSI standard is fully implemented in the R&S BASIC. There are additional graphics and I/O instructions which are similar in syntax to Microsoft BASIC. The instruction set also contains special functions such as IEC bus and interface instructions.

Just as any other programming language, the BASIC is first loaded into the main memory before it can be executed. Loading is carried out either

- by entering the command A>BASIC from the operating system, or
- by writing an AUTOEXEC file (BASIC is then automatically loaded together with the operating system).

When loading with the operating system, a BASIC program can already be specified and is then loaded together with the BASIC from the floppy disk or hard disk. This BASIC program is then automatically started by entering ,R (comma R).

Example `:E>basic test.bas,r`

As soon as BASIC has been loaded, it registers on the screen with BASIC, outputs the version number of BASIC and signals with READY that it is ready for further entries. It also labels the softkeys with commands. A small e behind the softkey labelling indicates that further data can be specified for this command and that it is only executed if the Return key is pressed. The other commands, e.g. RUN or CONT, are immediately executed when the softkey or function key is pressed. The softkey for EDIT mode is located on the left and is described in section 1.3.1.

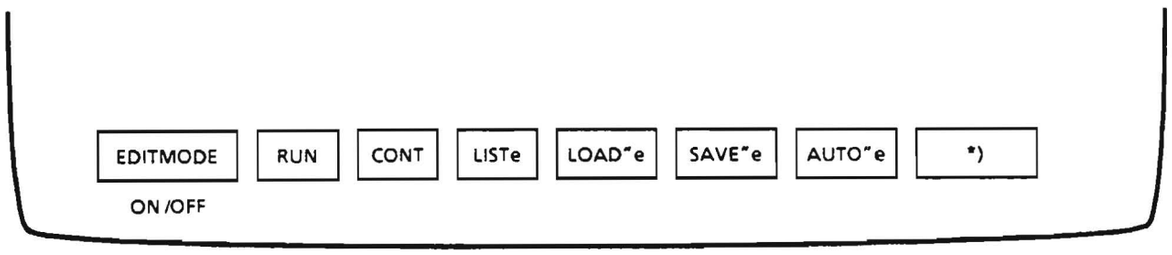


Fig. 1-1 Example of softkey labelling

*) for PSA/PAT: ALPH/GRA
for PCA: PRTSCRNv

1.2 Using the Keyboard in BASIC

The keyboard is used in BASIC to enter programs and to control a running program.

All keyboard functions are active in the BASIC editor. These keys can be divided into the following groups:

- Alphanumeric keys for program input
- Return and enter keys for program modifications
- Cursor keys and the rollkey for cursor movements
- Keys which modify the keyboard function
- Function keys which operate parallel to the softkeys in the BASIC editor
- Keys which control program execution
- Special controller keys.

1.2.1 Alphanumeric Keys

The alphanumeric keys are used to enter instructions, statements or numbers into the controller. The alphanumeric keypad also contains a series of special characters which mostly have a special function in the programs.

Upper-case or lower-case notation?

Entry of a program is considerably facilitated by the fact that the system accepts both upper-case and lower-case letters as being equivalent and that spaces can be expanded at will or omitted. After pressing the Enter key, the line is repeated in plain text as understood by BASIC, i.e. all key words are displayed in upper-case notation, whereas variable names and labels, except for the first letter, are written in lower-case. Spaces are inserted after key words in order to enhance the readability.

Note: If you are not quite sure about a keyword or the syntax, you should have another close look at the line when entered. If the system has converted the assumed keyword to lower-case letters, the controller has mistaken it for a variable or a subroutine call.

1.2.2 Enter Key

The Enter key terminates inputs to the controller

- entered into the running program using INPUT instructions
- following an instruction entry in direct mode. (This instruction is then immediately executed provided it is permitted in direct mode. An error message is otherwise output in the status line.)
- when editing a line where a line number is present at the beginning.

The line in which the cursor is currently located is transferred to the BASIC program if the Enter key is pressed during editing. If a line number already exists, the particular line will be deleted and replaced by the new line. It is unimportant whether the controller is in EDITMODE ON or OFF.

1.2.3 Rollkey

(only for PCA, EZM-B2, FS-K1 and PSA-Z1)

The function of the rollkey during editing is indicated by the LEDs above it. The function is identical with the corresponding individual cursor movements.

Direction of rotation	without Shift	with Shift
		

With the BASIC editor the horizontal shift cannot proceed beyond the start or end of the line. Vertical shifts beyond the upper edge of the screen enable up to 39 lines of the screen buffer to be displayed.

The rollkey changes its function with Shift and also with Ctrl. This function can also be read in by a BASIC program so that three independent, bidirectional ways of entry exist with the rollkey.

1.2.4 Cursor Keys

a) for PCA, EZM-B2 and FS-K1

The cursor function is active if the red LED of the Num Lock key is extinguished or if the cursor key is pressed together with SHIFT.

Home
Sets the cursor to the top left corner of the screen without deleting the screen.

Moves the cursor up by one position.

Moves the cursor one position to the left.
It is not possible to move beyond the leftedge of the screen. The cursor then remains at the first position of the line.

Moves the cursor one position to the right.
It is not possible to move beyond the right edge of the screen.

This key sets the cursor to the bottom left corner of the screen. The key is not labelled since there is no standard symbol for the function.

Moves the cursor down by one position.
Contrary to the LF key, a code which influences BASIC is not transmitted.

This key deletes the complete video memory, i.e. also characters present in the non-visible part, and sets the cursor to the top left corner.

Insert
The cursor changes its size when this key is pressed and indicates that the INSERT mode is active. In INSERT mode, characters can be inserted from the keyboard at the posiktion of the cursor.The characters to the right of the cursor are then shifted to the right. No further insertions are possible if the end of the line reaches the right edge of the screen. The INSERT mode is cancelled by pressing the key again, by pressing Del or a cursor key. The cursor is then of normal size again.

Delete
This key deletes the character at the cursor position. The characters in the same line to the right of the cursor are then shifted by one position to the left. It is possible to rapidly delete parts of a line by using this key together with the Repeat key.

Delete
This key in the alphanumeric keypad deletes the last entry, i.e. the last character before the cursor.

b) for PSA and PAT

The cursor function is active, if the green LED of the Num ↓ -key is extinguished or if the cursor key is pressed together with the Shift key.

Sets the curswor to the top left corner of the screen without deleting the screen.

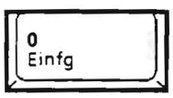
Moves the cursor up by one position.

Moves the cursor one position to the left. It is not possible to move beyond the left edge of the screen. The cursor then remains at the first position of the line.

Moves the cursor one position to the right.
It is not possible to move beyond the right edge of the screen.

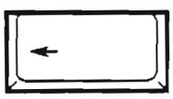
Sets the cursor to the bottom left corner of the screen.

Moves the cursor down by one position. Contrary to the LF key, a code which influences BASIC is not transmitted.



Einfügen
The cursor changes its size when this key is pressed and indicates that the EINFÜGE mode is active. In EINFÜGE mode, characters can be inserted from the keyboard at the position of the cursor. The characters to the right of the cursor are then shifted to the right. No further insertions are possible if the end of the line reaches the right edge of the screen. The Einfüge mode is cancelled by pressing the key again, by pressing Del or a cursor key. The cursor is then of normal size again.

This key deletes the character at the cursor position. The characters in the same line to the right of the cursor are then shifted to the left. It is possible to rapidly delete parts of a line by using this key together with the Repeat key.

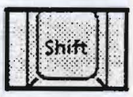


This key in the alphanumeric keypad deletes the last entry, i.e. the last character before the cursor.

1.2.5 Keys Changing the Keyboard Code

These keys cannot be read by the controller but they change the transmitted code of other keys.

a) for PCA, EZM-B2 and FS-K1



The two Shift keys are identical.
They change the function of most keys on the keyboard.

Upper-case letters are automatically selected on power-up or if the red LED on the key is lit. Lower-case letters are selected in conjunction with Shift. The LED is extinguished by pressing the Alpha-Lock key and lower-case letters are then active. Upper-case letters are then selected with Shift.

The input of BASIC programs may be in upper case or lower case. Lower-case letters are e.g. used in strings and texts. The Alpha-Lock key only affects the letters A to Z.

Cursor movements and editing commands can be entered using the keypad after power-up or if the red LED does not light up. The numbers can then be entered using Shift. Numbers can be entered by pressing the Num Lock key and the cursor movements can then be made together with the Shift key.

This key generates a control character with a code between 0 and 31 (decimal) as specified in the standards, if pressed together with another key of the alphanumeric keypad (A-Z @ ^ , . / -). These control functions are either immediately executed or displayed on the PCA screen with a special character. The control function is not affected, if the Shift key is pressed at the same time.

This key repeats another key pressed at the same time approx. 60 times per second. It is used for fast cursor movements or repeated entry of a character.

b) for PSA and PAT

The two Shift keys are identical.
They change the function of most keys on the keyboard.

Upper-case letters are automatically selected on power-up or if the green LED is lit. Lower-case letters are selected in conjunction with Shift. The LED is extinguished by pressing the Alpha-Lock key and lower-case letters are then active. Upper-case letters are then selected with Shift.

The input of BASIC programs may be in upper case or lower case. Lower-case letters are e.g. used in strings and texts. The Alpha-Lock key only affects the letters A to Z.

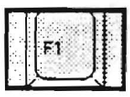
Cursor movements and editing commands can be entered using the keypad after power-up or if the red LED does not light up. The numbers can then be entered using Shift. Numbers can be entered by pressing the Num Lock key and the cursor movements can then be made together with the Shift key.

This key generates a control character with a code between 0 and 31 (decimal) as specified in the standards, if pressed together with another key of the alphanumeric keypad (A-Z @ ^ , . / -). These control functions are either immediately executed or displayed on the PCA screen with a special character. The control function is not affected, if the Shift key is pressed at the same time.

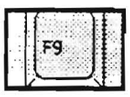
1.2.6 Function Keys

The function keys F1 to F8 trigger the same editing functions as the softkeys at the bottom of the screen. The associated text at the bottom of the screen is entered into the BASIC program just as if the text had been entered via the keyboard. The command is executed immediately if the text does not have a small e (extension); the text is written on the screen if there is an e. It can then be executed by pressing the Return key or extended by entering a statement such as LIST 100-200.

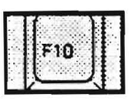
a) For PCA, EZM-B2 and FS-K1



EDIT-Mode



Insert Line
This key shifts the line with the cursor and all subsequent lines downward by one line; the last line on the screen is lost. This does not mean that the last line is no longer present in the program. This key can therefore be used to create space for the purpose of sending commands to the interpreter or editing a new line.



Delete Line
This key deletes the line with the cursor. The subsequent lines are shifted upward, and a blank line is entered into the last line.

b) For PSA and PAT

EDIT-Mode

This key switches between the alphanumeric and the graphics mode, independent of the labelling.

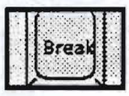
Insert Line
This key shifts the line with the cursor and all subsequent lines downward by one line; the last line on the screen is lost. This does not mean that the last line is no longer present in the program. This key can therefore be used to create space for the purpose of sending commands to the interpreter or editing a new line.

Delete Line
This key deletes the line with the cursor. The subsequent lines are shifted upward, and a blank line is entered into the last line.

1.2.7 Keys Controlling the Program Run

A running BASIC program can be accessed using these keys. Providing an INPUT or INKEY instruction is not present in the program, all other keys are ignored by the program and stored in a keyboard input buffer which has a capacity for 16 characters. These characters are output at the end of the program or upon a stop.

a) For PCA, EZM-B2 and FS-K1



A running BASIC program can be aborted using this key. Such an abort is only possible at the end of an instruction, since BASIC permits to read in variables, edit the program or continue the program run (CONT) following Break.The controller then indicates the line in which the program was aborted.

Control S can be used to stop outputs on the screen or printer, e.g. in order to examine them more closely. The program is continued when pressed a second time.

Using Control P, screen outputs can be simultaneously printed out on a printer.

b)For PSA and PAT

Untbr

A running BASIC program can be aborted using this key Such an abort is only possible at the end of an instruction, since BASIC permits to read in variables, edit the program or continue the program run (CONT) following Break.The controller then indicates the line in which the program was aborted.

Strg S can be used to stop outputs on the screen or printer, e.g. in order to examine them more closely. The program is continued when pressed a second time.

Using Strg P, screen outputs can be simultaneously printed out on a printer

1.2.8 Special Key

This special key is mainly used to enter strings with control functions.

Escape
The Esc key does not immediately trigger functions with the BASIC which means that it is possible to incorporate the Esc function into the program. Esc is absolutely essential when programming ANSI control functions as encountered e.g. with printers and plotters as well as in the softkey labelling of the PCA. The character Ec is output on the screen when the Esc key is pressed.

1.2.9 Using the Keyboard in a BASIC Program Run

In the BASIC program, all keys except the Break key can be read into the controller in one string using INKEY. Several codes are assigned to each key depending on the position of Shift, Ctrl, Alpha Lock and Num Lock and can be determined from the string using ASC(A$). The key code is then in decimal notation. A list of all key codes and their combinations in hexadecimal can be found in the Operating Manual of the controller.

The softkeys can also be read in using the INKEY instruction. Their function is independent of the keyboard. They occupy the codes A0H to A7H (hexadecimal), 160 to 167 (decimal)] where A0H is the softkey on the extreme right.

The rollkey has the following code assignment:
(decimal value in brackets)

	Direction of rotation	
	Backwards	Forwards
Without Shift	EA (234)	EB (235)
With Shift	CA (202)	CB (203)
With Ctrl	8A (138)	8B (139)

Note: Pressing a key on the keyboard always triggers an interrupt in the controller. If the statement ON KEY GOSUB n has been entered in the program before, a branch is made to the subroutine when the key is pressed, where the key code can then be processed further.

1.3 BASIC Editor

For the programming language BASIC, the capability of the editor is a vital factor in determining the development time of a software project. BASIC is a highly suitable interpreter for intermediate tests on incomplete software where rapid location and modification of instructions is particularly important.

The system has an extremely powerful video editor in which the usual disadvantages such as error messages in the listing, no backward scroll, transfer of unwanted lines into the program, string input mode etc. are avoided.

A screen editor is based on the device driver STRINX.SYS, which must be loaded with the file CONFIG.SYS for BASIC.

1.3.1 EDIT Mode

One of the exceptional features of the BASIC editor is the EDIT mode. In contrast to normal video editors which are always active except when the program is running, the EDIT mode permanently writes those lines which the user wishes to edit. In order to change or supplement several lines, the softkey EDIT mode on the left margin of the screen should always be used. EDIT mode is then indicated by ON and is switched off by pressing the softkey again or by RUN.

In addition to striking the softkey, a line number is required at the left edge of the screen in order to start the EDIT mode. This line number is either entered from the keyboard or generated by pressing the LIST softkey with a line number.

Once the EDIT mode has traced a line number in the program, it continuously builds up the previous or following line number using vertical cursor movements with the rollkey or the cursor keys.

Guidelines for particularly effective editing in the EDIT mode:

- In order to enter the line to be listed, it is sufficient to enter the approximate line number at the left edge of the screen. The line number range can then be displayed by vertical cursor movements using the rollkey or cursor keys.

- Line numbers can be written to the left edge of the screen irrespective of the text on the screen. Vertical cursor movements then display the new line number range on the screen. EDIT mode is then continued with this line number. A return to the old line number range can be made using ↖ or ↙ .

Caution! ***By pressing the Return key, the new line is transferred into the BASIC program memory.***

- A line is duplicated by assigning it a new line number and by pressing the Return key.

- Cursor movements outside the line range of the listing produce empty lines.

- Empty lines for inserting program lines can be produced using F9 (INSERT LINE). Lines can also be inserted by overwriting a line with a new line number and new contents and then pressing the Return key.

- Far-off line numbers can be reached in different ways:

  with Shift and rollkey
  with Rept and cursor
  by entering a new line number and moving the cursor
  by F9 (INSERT LINE) and entering a new line number and then moving the cursor.

- The complete alphanumeric screen can be cleared using the Clr (Stgr) key. A new line number must then be entered again and the cursor moved in order to produce a line number range.

- In order to test the program, simply press the RUN softkey. The LIST mode is then automatically switched off in order to be able to bring the last 30 lines back to the screen at the end or in the event of an interruption.

1.3.2 Using the ANSI Function under BASIC

The system possesses an internationally standardized software interface based on the ANSI standard X 3.41-1974 for addressing the screen and for cursor movements.

All functions required to operate a terminal are defined in this standard. These functions constitute the interface between BASIC and the screen output of ASCII characters.

The ANSI functions are called in BASIC using PRINT instructions. All functions start with ESC which is entered using the corresponding key on the keyboard and is displayed on the screen.

The following function groups are supported:

- Cursor control
- Labelling the status lines and softkeys
- Clearing the screen
- Setting the video attributes
- Output of display on printer

The parameters of the ESC sequences P_1 , P_2 to P_n are numbers and can be entered with 1 or 3 digits.

The available screen functions are described in the manual "Operating System for PCA" or in the manual of the option concerned.

1.3.3 Labelling the Softkeys in the Program Run

The labelling of the softkeys with BASIC commands is usually retained after starting the program. The labelling can be changed, however, depending on the program function. The labelling in the running BASIC program has no influence on the code of the softkey read in by the INKEY instruction.

Sequence: `ESC R` P_n `TEXT CR LF`

P_n: Number of the softkey 1 to 8 (from left to right)

TEXT: Text with up to 8 characters.

Function: The associated softkey is labelled with TEXT according to parameter P_n.

Example: Labelling of the 3rd softkey (from the left)

```
PRINT"Ec R3PROCESS"
```

Note: The sequence must be terminated by CR, LF.

If the automatic output of CR, LF is to be suppressed after pressing the key, the text must be terminated with 'e'.

The softkey must be terminated by "v" if the text is not to be output on the screen.

1.3.4 Labelling the Softkeys for Editing

If required, frequently used commands can be labelled on the softkeys for editing purposes. The command normally labelled on the softkey is then overwritten. This makes the editor even more effective.

The brightness of the softkey displayed in inverted form can be varied by setting a brightness level before.

Sequence: `ESC R` P_n `TEXT (e) [CR] LF`

P_n: Number of the softkey 1 to 8

TEXT: Instruction with up to 8 characters (including e)

Function: The associated instruction is written on the screen when the softkey is pressed. The PCA waits for a more detailed specification of the instruction if a small e follows. The instruction is immediately executed if not followed by a small e.

Example: `? "E$_c$R4RUN 100"`

The program is started in line 100 when softkey 4 is pressed.

Note: An incorrect notation or non-executable instructions lead to corresponding error messages as in the case of a direct input.

The labelling of the softkey on the outer left is not output when pressed.

Note: If the softkey labelling has been deleted in the program run or changed when editing, the original labelling can be restored by entering the SOFTKEY command.

Example:
```
SOFTKEY
```

1.3.5 Labelling the Status Lines

The system has a divided screen. The upper 25 lines are shifted, whereas the remaining lines on the screen remain stationary. These stationary lines can only be used for status information. All five lines can be labelled with 80 characters each.

Sequence: `ESC Q P$_n$ TEXT [CR] LF`

P_n: Number of status line 1 to 5
Text: Text with up to 80 characters

Function: The associated status line is labelled with TEXT according to parameter Pn.

Example: Labelling the first status line

```
? "EcQ1        DO NOT PRESS BREAK!"
```

Note: The second status line is overwritten in the case of error messages. The status line and the softkey labelling are cleared by ESC[y.

The sequence must be terminated by CR, LF.

Example: `? "Ec[y"`

Note: The attribute of the status lines can be set before labelling.

Example: `? "Ec[92,m";"EcQ5DARK ROW NR.5"`

1.3.6 Output of a Hardcopy on the Printer

The contents of the currently displayed ASCII video memory can be output on a printer connected to the Centronics interface by pressing the softkey or the function key on the printer. The softkey on the outer right of the PCA is preset with PRTSCRNv after loading BASIC.

Sequence: `ESC R Pn PRTSCRNv`

P_n: Number of the softkey 1 to 8

Function: The contents of the displayed ASCII video memory are copied via interface LPT1 by pressing the softkey labelled PRTSCRNv or the associated function key.

Example: `? "EcR4PRTSCRNv"`

Pressing the "Druck" key on the PSA and PAT controllers causes the content of the ASCII screen to be sent to a printer compatible to industrial standard.

In the program run, an ASCII hardcopy can be produced by outputting "ESC[z".

Sequence: `ESC[z`

Example: `1000 PRINT "Ec[z";`

Note: The output is optimized and outputs LF for blank lines. Special characters, e.g. B_L, cause the printer connected to act accordingly. A hardcopy of the graphics display is made using the instruction COPY OUT or GSAVE for controllers compatible to industrial standard.

1.4 Processing of Numbers and Variables in BASIC

1.4.1 Constants

Constants are generally decimal numbers between

$$1.7 \times 10^{308} \text{ and } 2.2 \times 10^{-308}.$$

Up to 16 places are possible and can be used for calculations. In accordance with the American notation, the decimal point is used instead of the comma.

Example: `25 ; 90.1234`

Constants can also contain exponents to base 10 which are identified by an E.

Example: `2 ▮ 103 ; 140.25 ▮ 10-12`

Exponential form: `2E3 ; 140.25E-12`

Since BASIC can process character strings in addition to numbers, string constants can also be used.

Example: "MEASURED VALUE A , "

String constants are marked by inserting them between quotation marks.

The alphanumeric display of the PSA and PAT controllers can be output on a printer compatible to industrial standard by pressing the print and the Strg key together.

1.4.2 Variables

The variable names can consist of a sequence of any length made up of letters, digits and the underscore character in order to arrange a program clearly. The first character must be a letter. When assigning names, ensure that the variable is not identical with the BASIC keywords since this would produce an error message.

Examples of permissible names:

```
100 K = 1
110 K1 = 1
120 Center = K1
130 First_K1 = 100
```

The following line produces an error message, because the word INPUT has already been reserved for an instruction:

```
150 input = 100     →     100 INPUT =
                                  ↑ Syntax Error
```

If it is absolutely necessary to use this variable name, the LET instruction may prove helpful in this case:

```
150 let input = 100    →    150 LET input = 100
```

(see also 1.2.1 Upper-case or lower-case notation).

1.4.3 Types of Variable

The computer differentiates between three different types of variable.

1.4.3.1 Floating-point Variables

The numeric variables may assume all values within the controller range of ± 1.797693134862 E + 308. Larger numbers produce an OVERFLOW ERROR. The smallest representable number is ± 2.225073858507 E-308. Smaller numbers are set to 0. Values within these limits can be assigned to the floating-point variables. The arithmetic accuracy using floating-point variables is exact to 16 decimal places, 13 of which are output.

Example:

```
A=7
Fd=1.5E6
X1=-0.2187E9
```

1.4.3.2 Integer Variables

These variables can be assigned integers up to + 32767 and –32768. They are identified by a % symbol at the end of the name.

Integer variables are represented internally as 16 bit numbers in twos complement form.

Values with a floating point can also be used, but the controller only uses the integral value; the number is always rounded off.

Example:
```
A%=7
Constant_2%=4528
X0%=0
```

Numbers within the range 32767 to 65536 are also accepted. For the output, however, a type of representation has to be chosen, and the numbers are then represented withing the range –1 to –32768 in this case.

```
20 B%=40000
30 B=B%
40 IF SGN(B)=-1 THEN B=65536+B
50 PRINT B%,B
```

→ –25536 40000

In line 40 of the example, an offset of 65536 is added to the value, if it is negative, in order to shift the number into the range 32767 to 65536.

1.4.3.3 String Variables

The string variables comprise a string with up to 65536 optional characters. They are identified by a $ character at the end of the name. The assigned character string constant is present within quotation marks which do not belong to the string, however, and are therefore not output e.g. with a PRINT instruction.

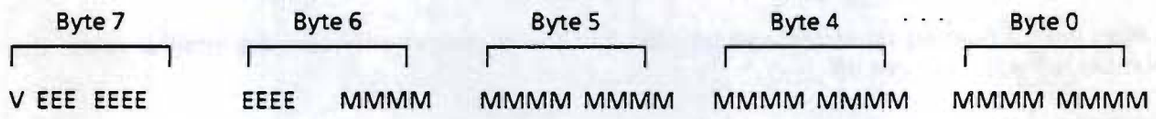
Example:
```
A$="TEXT"
Value$="55"
N1$=""
```

A special feature is the so-called null character string (N1$ in the example) which can consist of no characters.

Strings can be linked using the " + " character.

1.4.4 Internal Representation of Floating-point Variables

BASIC operates with the standard IEEE format with double precision. A floating point variable comprises 8 byte or 64 bit, of which 52 are the mantissa, 11 the exponent and 1 the sign.



V = Sign
E = Exponent
M = Mantissa

The exponent in the E-field is specified as a two's complement of the basic value 1024.

The mantissa is normalized, i.e. the MSB is always assumed to be "1" and is therefore not explicitly stored. An effective accuracy of 53 bit is therefore attained. The decimal value of the floating point variables described above is obtained by multiplying the mantissa by 2 ↑ (E-1023). It must be ensured that the MSB (bit 53) of the mantissa is always 1, i.e. the mantissa value can only be $1.0 \leq M < 2.0$. The 8 bytes are arranged in the memory in ascending sequence, i.e. the first byte 0 contains the LSB of the mantissa, the last byte, i.e. byte 7, contains the exponent and sign.

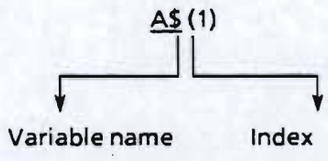
1.4.4.1 Internal Representation of Strings

First, the string identifier is saved. It consists of 4 bytes, the first word indicating the current string length and the second word its address (offset in data segment). This is the address where the string is located. A word indicating the maximum length of the string is located two bytes ahead.

1.4.4.2 Indexed Variables, Arrays

The arrays are a further group of variables. All three types of variable already mentioned are permissible. An index is added to the name of the variable, hence the term indexed variable. The index is written in brackets.

Example:



The variable name A$ remains the same for all elements in this field. Only the index varies. The index can also be a numerical expression.

Multi-dimensional fields, i.e. fields with more than one index, can also be produced. These indices are separated within the brackets by commas.

Example: Two-dimensional variable:

```
A(4,6) , Cx%(8,3) , String_A$(8,3)
```

Example: Three-dimensional variable:

```
X(3,2,4) , Fa%(4,0,3) , Gx$(3,4,2)
```

All numbers ≥ 0, numerical variables or numerical expressions can be used as indices for the variables. The variables can then also be indirectly addressed.

Example: Indirect addressing:

```
100 FOR I=0 TO 2
110 FOR J=0 TO 3
120 FOR K=0 TO 2
130 A(I,J,K)= RND(1)
140 NEXT K: NEXT J: NEXT I
```

A location for the indexed variable must be reserved using DIM before it can be used. A location comprises four bytes for a character string (only length and pointer; the character string is located at the upper end of the data segment), 2 bytes for an integer constant and 8 bytes for a floating decimal point variable.

Caution: Free locations are taken up unnecessarily by fields dimensioned too large.

1.4.6 Rules for Evaluating Expressions

The processing priority for mathematical expressions can be represented in a very simple way. The functions always have the highest priority after expressions in brackets, followed by the arithmetic and relational operators and the logical operators.

The priority is as follows:

Mathematical operators		Highest priority
Functions (SIN, LOG, etc.)		↓
↑		
*, /		
+, –		
Invert. (e.g. –SIN)		
=	Relational operators	
<> (or ><)		
>		
<		
>=		
<=		
NOT	Logical operators	
AND		
OR		
XOR		Lowest priority

Example: ? 2 ↑ 3 + 6 results in 14, since 2 ↑ 3 is calculated first.

1.4.7 Special Features of the Equal Sign

The equal sign does not have the same meaning as it has in algebra.

It means: "Receives the value of"

and must not be confused with the algebraic meaning:
"Is equal to".

The statement X = X + 1

would be algebraically incorrect since it contains the contradiction 0 = 1. This statement is correct in BASIC, however, and means:

X "receives the value of" the old value of X plus 1

or

X = X · B

X "is the value of" the old value of X times the value of B.

(See also 1.4.9.1 Relational Operators)

1.4.8 Mathematical Expressions as Examples of Compute Statements

Mathematical expressions are written in BASIC in a manner slightly different from the common notation. The following list illustrates algebraic and BASIC notations.

Algebraic	BASIC
$\frac{a+b}{c+d}$	`(A + B) / (C + D)`
$\frac{a+b}{c}$	`(A + B) / C`
$\frac{a \times b}{c}$	`A · B / C`
$\frac{\frac{a}{b}}{c}$	`A / B / C`
$\frac{a}{\frac{b}{c}}$	`A / (B / C)`
$\frac{a}{b \times c}$	`A / (B * C)`

$(ab)^{NC}$	`(A * B) ↑ N * C`
$ba+1$	`B ↑ (A + 1)`
$ba+1$	`B ↑ A + 1`
$2\sqrt{a^3}$	`A ↑ 0.6` [3/5 = 0.6]
$\frac{1}{2\sqrt{a^1}}$	`A ↑ (-0.5)` [1/2 = 0.5] or 1/SQR (a)

The compute statement is then generated by the equal sign whose special feature has already been referred to. A compute statement is represented in the same manner as in algebra.

e. g. X = A + B + C

- A + B + C — Mathematical expression
- X — A variable to which the result is assigned.

1.4.9 Operators

1.4.9.1 Relational Operators

Relational operators are used to compare numerical values or strings. The result is 0 if the comparison is not fulfilled (false) or -1 if the condition is fulfilled (true). This means that all bits are 1 so that the comparisons using the Boolean operators can be linked further without limitations.

In the case of larger/smaller comparisons of strings, these are processed character-by-character starting at the left. The first character of the two strings found not to be identical is used for the comparison. The ASCII value of the characters are compared. This leads to the result of the character string comparison.

>			Greater than
<			Less than
> =	or	= >	Greater than or equal to
< =	or	= <	Less than or equal to
< >	or	> <	Not equal to
=			Equal to

Examples of comparisons:

```
100 IF A<B THEN ?A

200 IF C>=A*B THEN ?C

300 IF A<>100 THEN 1000

400 IF A<100 AND A>10 THEN GOTO 20
```

Example of the comparison of strings:

```
400 IF A$="E" THEN ?"READY"
```

1.4.9.2 Logical Operators

AND

OR

NOT

XOR

Boolean operators represent an indispensable aid in the digital and control fields whenever it is necessary to link individual bits of binary numbers. They are also often useful for complex read-in tasks.

Syntax:

```
a AND b
a OR  b
NOT a
a XOR b
```

a,b: Constant, variable or numerical expression; places after the decimal point are rounded off.

The Boolean operators are used to link numbers bit by bit. These operators can be used without limitation in a mathematical expression. The numbers used are first converted from the floating-point format to a 16-bit number in the range between -32768 and 32767. Range violations cause an error message. Negative numbers are represented in two's complement (with leading 1). Places after the decimal point are rounded off.

Example of logical operations

```
10 LET A = 5 OR 12                0101 ≙ 5
                                OR 1100 ≙ 12
           →A = 13                1101 ≙ 13

20 LET B = A AND 6                01101 ≙ 13
                                AND 00110 ≙ 6
           →B = 4                 00100 ≙ 4

30 LET C = NOT A                    A                          1101 ≙ 13
           →C = -14           NOT A                1111111111110010 ≙ -14

40 LET D = 7 XOR A                00111 ≙ 7
                                XOR 01101 ≙13
           →D = 10                01010 ≙ 10
```

Example:

```
100 C=G AND 16
200 IF A=5 OR C=3 THEN500
300 H=NOT H
400 X=(A OR B)AND NOT C
```

Note: A comparison provides the value -1(all the 16 bits are set) if it is "true".

1.4.9.3 Power Operator

a ↑ b This generates the power function with any base and any exponent.

Mathematical: $Y = a^b$

Syntax: `a↑b`

a,b: Constant, variable or numerical expression

Example:

```
100 Y=A↑X
```

1.5 Using the IEC Bus under BASIC

1.5.1 Introduction into the IEC-bus Syntax

Note: The hardware of the IEC bus with extracts from the standard can be found in the operating manual "Process Controller".

The controller has its own instruction set for operations with the IEC bus since it is an interface with a large number of functions. A common feature of all instructions is that they commence with IEC and can therefore be readily distinguished from the remaining BASIC program. All instructions can be written without further additions (e.g. IEC OUT10,A$) if only one IEC bus is fitted in the PCA. If a second IEC bus is to be accessed, the number of the IEC bus is specified after IEC (e.g. IEC 2OUT 10,A$). If IEC is followed by a 1 or by no data at all, the instruction is executed for the 1st bus. All instructions, i.e. also IECTERM, IECTIME, IECRLC etc., are only executed for the specified bus and must be specified again for the second IEC bus if required.

If several IEC instructions are in a sequence, the letters IEC need only be specified in the first instruction. All following statements are separated by commas and joined to the first one (without IEC).

Example: 2 IEC buses, one of which transfers the control

```
100 IEC TERM13, TIME100, EOI, LAD12, $OUT"TEST"
110 IEC 2RLC, ADR7, TIME20000, TERM 0
```

The IEC-bus instruction set comprises the following types of instruction:

a) Complex instructions

```
IEC OUTa,
IEC INa,
```

These are normally sufficient to control measuring instruments so that no further instructions need be specified. They address the instrument specified in a, handle the data transfer and unaddress the instrument again.

b) Instructions to set the interface, e.g. delimiter or time out.

```
IEC TERM    IEC ADR
IEC TIME    IEC T1
```

Need only be specified if the default values are not used.

c) Instructions for data input and output

```
IEC %IN     IEC %OUT
IEC $IN     IEC $OUT
```

d) **Instructions for transmitting IEC commands and statements.**

```
IEC LAD     IEC LLO
IEC TAD     IEC DCL
IEC SAD     IEC PPU
IEC MLA     IEC MTA
IEC GTL     IEC SPE
IEC SDC     IEC SPD
IEC PPC     IEC PPD
IEC GET
IEC TCT
```

A data word is output on the IEC bus together with ATN.

e) **Instructions for transmitting line messages.**

```
IEC ATN     IEC NATN
IEC IFC
IEC REN     IEC NREN
IEC EOI     IEC NEOI
```

These influence the lines of the management bus.

f) **Instructions to carry out the serial poll or parallel poll.**
(Some of these instructions are also encountered under point d).

```
IEC PPC     IEC SPL
IEC PPE     IEC SPE
IEC PPL
IEC PPD     IEC SPD
IEC PPU
```

g) **Instructions required when transferring the control in the presence of several controllers:**

```
IEC RLC
IEC TCT
IEC WTCT
IEC WMLA
IEC WMTA
```

1.5.2 Several Controllers on the Bus

The computer is usually the only controller on the bus. It then performs the system control and can transmit the messages IFC and REN. The jumper X5 must be plugged in between pin 1 and pin 2. In the PSA/PAT controllers the signal direction of these control lines is automatically reversed.The message IFC is automatically output on the bus when starting BASIC with RUN.

If two IEC buses are installed in the controller, they both usually perform the system control. IFC is then transmitted on both buses when BASIC is started. Following the start, the system controller also holds the lines ATN and REN active low (true).

If two controllers are connected to ***one*** IEC bus, one of them must relinquish the system control and become the addressed controller. In the PCA jumper X5 of the addressed controller must then be inserted between pin 2 and pin 3. After starting BASIC, the control on the bus must be passed back with IECRLC as the first instruction so that the IEC instructions are also initialized accordingly. IFC and REN must not be transmitted after IECRLC.

Following IECRLC, the computer is first present as a talker/listener on the bus. In this function it must not set ATN to true, i.e. it must not transmit addresses and commands. An attempt leads to the error message "*not a IEC-Bus-Controller*".

If the computer serves as talker/listener, BASIC requires a bus address. It is assigned the address using IECADR. The address may be changed while the program is running.

Data transfer is then performed via the instructions IEC%IN and IEC$IN or IEC%OUT and IEC$OUT. BASIC executes these instructions only if it has been correctly addressed before (i.e. as a talker or listener). After unaddressing the controller, a bus transfer between the system controller and other devices on the bus can be handled without these data being read into the IN instruction, for instance.

The program is aborted with a TIME OUT error message if the controller has not been addressed within the time specified by IECTIME and if data transfer has not commenced.

A further possibility of synchronizing data transfer is provided by the instructions IECWMLA and IECWMTA. At this point, the computer waits for being addressed. These instructions do not allow to abort the program with TIME OUT in order to be able to synchronize rare and irregular data transfer as well.

The computer can of course also assume the control (not the system control) on the bus. It waits for taking over the control with the instruction IECWTCT. After applying its talker address to the bus and receiving the message TCT, it executes the IECWTCT instruction and then possesses the control on the bus. It can now transmit addresses and commands (ATN true) and thus control the connected devices. Control is released again by transmitting the talker address of the controller which is to take over the control and the message IECTCT. The controller then serves again as talker/listener on the bus.

1.5.3 Using the Line Message Service Request

1.5.3.1 The Computer as Controller

As a controller, the computer can receive a service request. If a service request appears on the bus, i.e. if the SRQ line is set to low, the BASIC program can branch to a subroutine if the statement ON SRQ GOSUB m or ON SRQ GOTO m has been executed before. If an SRQ is present on the bus, BASIC completes the current instruction and then executes the first instruction of the subroutine or the branch destination m. A serial poll should then be carried out in the subroutine or at the branch destination since most devices do not enable the SRQ line again before their status has been read in by a poll.

Application: In particular if the devices connected have long response times (>100 ms), the execution time of a BASIC program can be reduced by fetching the measured value in a service request routine. If several controllers are connected to the bus, they can be serviced in a poll routine and the control can be transferred to another controller if applicable.

1.5.3.2 The Computer as Talker/Listener

The device can transmit service requests as a talker/listener, i.e. if the control has been transferred via the bus using IECRLC. This is carried out using the instruction IECRQS b. When the instruction is executed, the SRQ line is first set to low and the program is continued. After receiving the message Serial Poll Enable and being addressed with the talker address defined in IECADR, BASIC applies the status byte b (decimal 0 to 255) defined in the IECRQS instruction to the bus. Bit 7 (decimal 64) of the status byte is high as long as the SRQ is present. The computer cancels the service request only after this byte has been fetched by the controller. The SRQ line then returns to high, provided that there is no service request from another device.

The computer has the capability of participating in a serial poll. The status word applied in the serial poll depends on its status.

0 after IECRLC
64 OR b after IECRQS b (bit 7 is set)
191 AND b after IECRQS b and cancelling of the SRQ line (bit 7 is reset)

1.5.4 Execution of Parallel and Serial Polls

A good method of avoiding the occurrence of waiting times for measured values from IEC bus devices is to read in the values in a service request subroutine. BASIC executes the program and branches into a subroutine upon occurrence of a service request in which the device can then be serviced. A prerequisite is that the instrument has a service request facility and that the statement ON SRQ GOSUB has been executed (see Section 2.3).

If several devices can transmit SRQ on the bus, the instrument raising the service request must be determined at the start of the SRQ subroutine.

This can be carried out either sequentially in a serial poll or simultaneously in a parallel poll.

1.5.4.1 Serial Poll

The easiest method of detection is a serial poll with the statement IECSPL b,v%. All devices from which the service request might originate are then addressed one after the other with their address b and their device status is read into the variable v% as an 8-bit integer.

Example:

```
100 ON SRQ GOSUB 1000
110 Program
.
.
.
1000 IEC SPL 5, A%
1010 IEC SPL 12, B%
1020 IEC SPL 14, C%
1030 REM Processing of A%, B%, C%
.
.
.
1100 ON SRQ GOSUB 1000: RETURN
```

As is the case with all combined IEC-bus instructions, the serial poll can also be executed using individual instructions.

The instruction IECSPL b, v% corresponds to the following individual statements:

Example:

```
IEC SPE
IEC TAD b
IEC %IN A%
IEC MTA
IEC SPD
```

Unaddressing is not necessary between the polls of several instruments. The first example can therefore also be implemented with a smaller number of bus cycles as shown in the following:

Example:

```
1000 IEC SPE,TAD5,%INA%,TAD12,%INB%,TAD14,%INC%
1010 IEC MTA,SPD
```

1.5.4.2 Parallel Poll

The parallel poll is the second method which allows the controller to poll the status of the SRQ line of the connected devices. The parallel poll sets the lines EOI and ATN to low and reads in the status word applied to the data bus by the devices. The parallel poll is thus carried out faster than the serial poll. Prior to polling, each device is assigned its own data line (DIO) on which it can signal its status. In contrast to the serial poll, the parallel poll does not change the SRQ status of the polled devices. A parallel poll can only be performed by the controller on the bus.

BASIC supports the parallel poll with a series of IEC instructions. The device setting is present in the main program since it need only be made once. Devices not set for parallel poll do not participate.

Example:

```
100 ON SRQ GOSUB 1000
110 IEC LAD 5,PPC ,PPE 1 6,UNL or:
110 IEC PCON 5,1,6
120 IEC LAD 12,PPC ,PPE 0 4,UNL or:
120 IEC PCON 12,0,4
```

The first parameter of the IECPPE instruction indicates whether the polled device signals its SRQ status with 0 or 1. The second parameter identifies the DIO line on which the reply is to be made.

The parallel poll is triggered by the IECPPL instruction.

Example:
```
1000 IECPPL A%
1010 IF A% AND 32 THEN ...
```

The status word is read into the variable A% during the parallel poll. Then the service request line and thus the device sending the SRQ can be determined. This device is then polled in a serial poll routine.

The controller can also participate in a parallel poll as a talker/listener. In this case it is necessary to assign to the controller one of the 8 parallel poll lines on which it is supposed to signal its status. This is carried out in the following steps:

1) Addressing the controller as listener
2) Transmitting the command Parallel Poll Configure
3) Transmitting the command Parallel Poll Enable with indication of the line on which the controller is to respond as well as indication of the sense bit 0 or 1
4) Unaddressing the controller

By simultaneously applying EOI and ATN, the controller can now be requested to apply its identification to the bus. It then applies its SRQ status to the bus line assigned to it under point 3 in true or inverted form depending on the sense bit.

Application: The computer can use a service request to inform the controller that a measured value or a calculated result is present or that it wants to take over again the control on the bus. Functions can be differentiated using the status word in the serial poll.

1.6 Incorporation of Assembler Subroutines in BASIC Programs

This section describes the incorporation of assembler subroutines in BASIC programs. In particular, information is provided on the following points:

- procedures by means of which user-specific subroutines can be loaded,
- handling of the interface between BASIC and the subroutines,
- examples of the procedures.

BASIC occupies approx. 160 Kbyte of free storage space joining onto MS-DOS. This block contains the BASIC interpreter (with data and stack) and the BASIC user program (with data).

A block within or outside the BASIC memory range can be assigned to the assembler subroutines by using different procedures.

We recommend the four procedures listed below for generating or loading assembler subroutines:

- in an integer matrix within the BASIC data segment
- with Poke within or outside the BASIC data segment
- with LOAD# und CALL# outside BASIC
- as a memory-resident part of DOS outside BASIC.

The following documents are recommended for the generation of assembler routines:

Microsoft MS-DOS User's Guide/User's Reference
Microsoft MS-DOS Programmer's Reference Manual

1.6.1 Procedure for Loading Subroutines

Assembler routines can be loaded according to one of the following procedures. The selection of the procedure depends on the size and the characteristics of the assembler program.

1.6.1.1 In an Integer Matrix within the BASIC Data Segment

A small subroutine can be easily initialized by loading into an integer matrix within the BASIC data segment.

Characteristics:

No assembler is required in the writing of programs.

The complete code (BASIC and machine code) is located within one file.

When loading the subroutine into a memory area under BASIC control, there is no danger of accessing critical areas of the memory.

Notes:

If the BASIC user program occupies the greater part of the data area, memory space is lost by assigning the subroutine to the data segment.

The machine code must be relocative. This means that it must not contain any variables whose values depend on the load address of the subroutine.

1.6.1.2 With POKE within or outside BASIC

This procedure is suitable like procedure 1 (Section 1.6.1.1) for small subroutines. The difference is that routines can also be filed outside the BASIC data segment.

Characteristics: see Section 1.6.1.1

Notes: see Section 1.6.1.1

The memory area must be managed by the user in order to protect the routines from DOS or other programs if the assembler subroutine is loaded in a memory area outside the BASIC data area.

1.6.1.3 Outside BASIC (with LOAD# and CALL#)

This procedure is suitable for programs up to 64 Kbyte. A vacant memory area outside BASIC is required. BASIC manages the available memory for the user in order to protect the user-specific subroutine.

Characteristics:

No memory space in the BASIC data segment need be used for the subroutines.

Notes:

The user is not responsible for managing the occupied memory area.

1.6.1.4 Outside BASIC as Memory-Resident Part of the DOS

This procedure can be used to load programs whose size is practically only limited by the memory space available.

Characteristics:

The procedure enables multi-segment routines to be used which cannot be handled using LOAD# and CALL#.

The procedure converts the subroutine into a memory-resident DOS part; it is not necessary to reserve a memory area.

Notes:

DOS-loaded subroutines remain memory-resident after loading. The routines are only cleared from memory when the operating system is booted again. Thus multiple copies can be resident in the memory. In order to clear the routine from the memory, a branch to the bootstrap loader can be made at the end of the BASIC program.

1.6.2 Interface between BASIC and Assembler Subroutines

The following applies if the user-specific subroutine is called:

- The DS register is set to the BASIC data segment.
- A stack area of 10 words is available for the routine. If a larger area is required by the subroutine, its own stack must be created where at least 128 byte are not used by the subroutine (interrupts, DOS).
- The address of the variables in the stack is transferred for each parameter in the call. The segment address and then the offset address of the variables are written onto the stack.

  If the transfer parameter is a string, the transfer address is a pointer at the string descriptor. The format is as follows:

  Word 0 contains the current length of the string.
  Word 1 contains the offset of the string.

  The word ahead of the string contains the maximum string length.

  Caution: The subroutine must not change the code for the maximum length of the string.

The following applies to the return to BASIC:

- Interrupts inhibited by the subroutine must be enabled.
- The registers SS and DS must contain the original values again.
- The actual return takes place using a far return.

Access to parameters in the stack:

The macro assembler contains the pseudo instruction STRUC in order to define e.g. offsets.

Example:

```
FRAME   STRUC

        RET_OFF             DW      ?
        RET_SEG             DW      ?

        PARA_N_OFS          DW      ?
        PARA_N_SEG          DW      ?
        .
        .
        .
        PARA_1_OFS          DW      ?
        PARA_1_SEG          DW      ?

        FRAME   ENDS
```

It is then possible to access the stack parameters in the following manner:

```
MOV     BP, SP
MOV     BX,[BP.PARA N OFS]      ;LOAD OFFSET OF PARAMETER N IN BX . . . .
.
.                               ;ETC.
.
.
```

1.6.3 Examples

Procedure 1 (procedure 2)
(Sections 1.6.1.1 and 1.6.1.2)

The machine code for the assembler subroutine is first determined (e.g. with Debug).

A sufficiently large integer matrix is dimensioned in the BASIC program.

The machine code is assigned word by word to the matrix elements. It must be taken into account that the processor first expects the least significant byte when reading the instructions. The data must therefore be stored in this sequence.

- The position of the subroutine within the BASIC data segment is determined using the VARPTR function.
- The subroutine can be called using the CALL instruction.

Example:

```
10 REM   ************************************************************
20 REM   *                                                          *
30 REM   *                  B1.BAS \ 31.03.85                       *
40 REM   *                                                          *
50 REM   * THE PROGRAM ENABLES INTERRUPT FLAGS TP BE SET AND        *
60 REM   * RESET                                                    *
70 REM   *                                                          *
80 REM   ************************************************************
90 REM
100 DIM AR%(2)
110 DATA "CBFA"
120 DATA "CBFB"
130 FOR I=0 TO 1: READ A$: AR%(I)=HEX(A$): NEXT I
140 STI=VARPTR(AR%(1))
150 CLI=VARPTR(AR%(0))
160 CALL STI
170 CALL CLI
180 END
```

Procedure 2
(Section 1.6.1.2)

The difference between procedure 2 and procedure 1 is that the bytes are stored in the matrix or outside the BASIC memory area using the POKE instruction. The POKE instruction itself ensures that the correct byte sequence is retained. This procedure is therefore more suitable than procedure 1 for small programs.

In order to set the subroutine outside the BASIC data area, the required segment address must be specified in the BASIC program using the SEGMENT instruction. References using VARPTR are then no longer possible because the subroutine is stored outside the BASIC data segment.

Example:

```
10 REM ************************************************************
20 REM *                                                          *
30 REM *                  B2.BAS\31.03.85                         *
40 REM *                                                          *
50 REM * IN THIS EXAMPLE, THE DOS FUNCTION 30H IS CALLED. THE     *
60 REM * DATA RECEIVED BY DOS ARE TRANSMITTED TO THE BASIC        *
70 REM * COMMUNICATION AREA (40:20), BASIC PROGRAMS CAN READ      *
75 REM * THESE DATA USING PEEK.                                   *
77 REM *                                                          *
80 REM ************************************************************
90 REM
100 DIM AR%(10)
105 REM
110 DATA "B4","30": REM                 MOV  AH,30H
120 DATA "CD","21": REM                 INT  21H
130 DATA "1E": REM                      PUSH DS
140 DATA "BB","40","00": REM            MOV  DX,40H
150 DATA "8E","DB": REM                 MOV  DS,DX
160 DATA "A3","20","00": REM            MOV  WORD PTR[20],AX
170 DATA "1F": REM                      POP  DS
180 DATA "CB"; REM                      RETF
185 REM
190 P=VARPTR(AR%(0)): FOR I=0 TO 14: READ A$: POKE (P+I),HEX(A$): NEXT I
200 SUBR=VARPTR(AR%(0)): CALL SUBR
210 SEGMENT HEX("40")
220 MA%=PEEK(HEX("20"))
230 MI%=PEEK(HEX("21"))
240 SEGMENT DEF
250 PRINT "MS-DOS ";MA%;".";MI%
260 END
```

Procedure 3
(Section 1.6.1.3)

In this procedure the instructions LOAD# and CALL# contained in the BASIC are used to directly load a "program file" from the floppy disk or hard disk into the memory. The .COM file is generated by assembling a source, linking the OBJ file, converting the EXE file and renaming the .BIN file.

Example:

```
1 REM  ************************************************************
2 REM  *                                                          *
3 REM  *               B3.BAS \ 31.03.85                          *
4 REM  *                                                          *
5 REM  *    THIS BASIC PROGRAM LOADS AND CALLS A .COM FILE        *
6 REM  *                                                          *
7 REM  *                                                          *
8 REM  *                                                          *
9 REM  ************************************************************
10 REM
15 LOAD# 1,"B3.COM"
20 CALL# 1,P1%,P2%,P3%,P4%
30 IF P1%=HEX("FFFF") THEN80
40 PRINT "SECTORS PER CLUSTER   :";P1%
50 PRINT "AVAILABLE CLUSTERS    :";P2%
60 PRINT "BYTES PER SECTOR      :";P3%
70 PRINT "CLUSTERS PER DRIVE    :";P4%
80 END
```

```
;       B3.ASM
;
;
;       THIS EXAMPLE SHOWS HOW AN ASSEMBLER SUBROUTINE DETERMINES
;       THE SECTORS PER CLUSTER, AVAILABLE CLUSTERS, BYTES PER
        SECTOR AND TOTAL CLUSTERS PER DRIVE USING THE DOS CALL 36H
        AND RETURNS THESE DATA TO THE BASIC VARIABLES.
;
;
;
;       GENERATION OF THE .COM FILE:
;
;       - CREATION OF THE SOURCE FILE (B3.ASM) WITH AN EDITOR
;         (E.G. WORDSTAR)
;
;       - MASM B3               (ASSEMBLE SOURCE)
;       - LINK B3               (LINK .OBJ FILE)
;       - EXE2BIN B3            (CONVERT .EXE / .BIN)
;       - COPY B3.BIN B3.COM    (GENERATE .COM FILES)


PAGE 69,132                     ;LISTING DIRECTIVE

PARSTRUC      STRUC             ;PARAMETER LIST
;
OFS_RET DW    0                 ;OFFSET FOR RETURN
SEG_REG DW    0                 ;SEGMENT
;
OFS_P4  DW    0                 ;OFFSET PARAMETER 4
SEG4    DW    0                 ;SEGM.  "       " "
OFS_P3  DW    0                 ;OFFSET PARAMETER 3
SEG3    DW    0                 ;SEGM.  "       " "
OFS_P2  DW    0                 ;OFFSET PARAMETER 2
SEG2    DW    0                 ;SEGM.  "       " "
OFS_P1  DW    0                 ;OFFSET PARAMETER 1
SEG1    DW    0                 ;SEGM.  "       " "
;
PARSTRUC      ENDS              ;END OF PARAMETER LIST


CODE    SEGMENT 'CODE'          ;SEGMENT = CODE ; CLASS = CODE
        ASSUME CS:CODE,DS:CODE  ;CODE AND DATA IN CODE SEGMENT
        ORG     100H            ;ADRESS LEVEL = 100H
                                 (required!)

SUBR    PROC    FAR             ;DECLARATION OF A FAR PROCEDURE
STRT:   JMP     BEGIN           ;JUMP TO START OF CODE

WORD1   DW      ?                ;USER DATA DEFINITIONS
(IM CODE)
WORD2   DW      ?                ;
WORD3   DW      ?                ;(NOT USED IN EXAMPLE)
WORDN   DW      ?                ;END OF USER DATA


BEGIN:  MOV     DL,0             ;DEFAULT DRIVE
```

```
        MOV     AH,36H              ;FUNCTION 36H
        INT     21H                 ;ENTRY INTO DOS

        MOV     BP,SP               ;BASE POINTER ON STACK
        MOV     DI,[BP.OFS_P1]      ;OFFSET BASIC-PARA 1
        MOV     [DI],AX             ;IN BASIC-PARAMETER 1
        MOV     DI,[BP.OFS_P2]      ;OFFSET PARA 2
        MOV     [DI],BX             ;IN BASIC-PARAMETER 2
        MOV     DI,[BP.OFS_P3]      ;OFFSET PARA 3
        MOV     [DI],CX             ;IN BASIC-PARAMETER 3
        MOV     DI,[BP.OFS_P4]      ;OFFSET PARA 4
        MOV     [DI],DX             ;IN BASIC-PARAMETER 4
        RET

SUBR    ENDP                        ;
CODE    ENDS                        ;
END     STRT                        ;(required!)
```

Procedure 4
(Section 1.6.1.4)

The EXE program which can be directly executed by the operating system is loaded into the memory by entering the name.

The loaded program must consist of two parts, the load part and the actual subroutine.

The following steps must be carried out in the load part:

- The entry address of the subroutine must be filed in the BASIC user communication area.
- The DOS must be informed of which part of the program is to remain memory-resident (see operating system function 31H).

The memory area provided for data exchange between BASIC and assembler subroutines is located in segment 40H at offset 20H and is 4 byte long. The load part files the segment and offset addresses of the entry point of the actual routine here.

Example:

```
10 REM  ************************************************************
20 REM  *                                                          *
30 REM  *                B4.BAS \ 31.03.85                         *
40 REM  *                                                          *
50 REM  *  THIS BASIC PROGRAM CALLS A RESIDENT SUBROUTINE          *
60 REM  *                                                          *
70 REM  *                                                          *
80 REM  *                                                          *
90 REM  ************************************************************
100 SEGMENT HEX("40")
110 OF=PEEK(HEX("20"))+256*(PEEK(HEX("21")))
120 SG=PEEK(HEX("22"))+256*(PEEK(HEX("23")))
130 SEGMENT SG
140 CALL OF
```

```
;B4.ASM
;
;       THIS ASSEMBLER SUBROUTINE CONTAINS TWO PARTS: ;
;       - THE LOAD PART
;       - THE ACTUAL SUBROUTINE
;
;       THIS PROGRAM ONLY OUTPUTS TWO CHARACTERS TO INDICATE THAT
;       IT HAS BEEN CALLED CORRECTLY BY BASIC.
 PAGE 69,132                            ;LISTING DIRECTIVE

PROGSIZE EQU
((OFFSET ENDE - OFFSET START +100H)/16) + 1 ;SIZE IN PARAGRAPHS:
                                        ;OF RESIDENT PORTION
                                        ;OF USER PROGRAM

CODE    SEGMENT 'CODE'
        ASSUME CS:CODE
```

```
SUBR    PROC    FAR
START:  JMP     BEGIN

WORD1   DW      ?
WORD2   DW      ?
WORD3   DW      ?

BEGIN:  MOV     DL,'O'
        MOV     AH,2
        INT     21H
        MOV     DL,'K'
        MOV     AH,2
        INT     21H
RET
SUBR    ENDP

END:
LOADER  PROC
        MOV     AX,40H
        MOV     ES,AX
        MOV     AX,OFFSET SUBR
        MOV     ES:[20H],AX
        MOV     AX,CS
        MOV     ES:[22H],AX
        MOV     DX,PROGSIZE
        MOV     AH,31H
        INT     21H

LOADER  ENDP
CODE    ENDS
END     LOADER
```

1.7 Using Files and Interfaces

It is possible to read data from or write data to a file on the floppy disk or hard disk. This also applies to hardware interfaces such as printers, IEC bus, serial communications interface, and so on. This is why the operating system and BASIC do not always make a distinction between them; the same input/output instructions such as PRINT#, INPUT#, INPUT$ are used for both files and interfaces. The OPEN instruction is used to determine the interface or file to be prepared first.

There are two OPEN instructions:

OPENO opens a file for the output, i.e. an already existing file is overwritten, or a new file is created under the indicated name.

OPENI opens a file for subsequent inputs. If this file does not exist, an error message is produced.

This is illustrated by the following example (more details will be given at a later date).

```
100 OPENO#1,"DATEI.DAT"
110 PRINT#1,TIME$
120 CLOSE#1
```

A file with the name "DATEI.DAT"is opened and the current time recorded in it. The time stored can be read by opening the file:

```
300 OPENI#13,"DATEI.DAT"
310 INPUT#13,A$:PRINT A$
120 CLOSE#13
```

The filename can be a sequence of max. 8 characters before and max. 3 characters after the point. The permissible characters can be found in the manual "MS-DOS User's Guide / User's Reference". This manual explains how to select a drive by placing a letter ahead with a colon and how to use a sub-directory separated by backward slashes if the paths preset by MS-DOS are not to be used.

Both upper-case and lower-case letters can be used.

Example of a permissible name:

```
100 OPENO#1,"E:\User\Datei.dat"
```

In MS-DOS, a few names are allocated to interfaces and cannot be used as filenames:

AUX, CON, LST, PRN, NUL, CLOCK

Other names are reserved for optional device drivers*[1] and are allocated as soon as the associated device driver is loaded with CONFIG.SYS:

STRING, COM1, COM2...6, IEC, BEEPER, TTL GRAPH, ANG, LPT1, LPT2, UCI, DOP, ADC, REL1...4

If the reserved names listed above are used, MS-DOS attempts to address the associated interface. This is also the case if these names have a file extension (after the point)!

*) A device driver is a software module which makes the hardware accessible via a standardized software interface.

In the example of the OPEN instruction, a # character is used, followed by a number. This number is referred to as the channel number and establishes the reference between the OPEN instruction and the subsequent input or output instructions. It can be freely selected by the user in the range between 1 and 15. Of course, a particular channel number can be allocated only once at a time. How many channels can be open at a time is a function of many factors:

The file CONFIG.SYS with FILES = n informs MS-DOS about how many files are to be open at a time (default 8) (see Appendix D of the MS-DOS User's Guide). MS-DOS provides max. 20 files to be opened for each process (BASIC is a process in this case). Five of them are already reserved for standard inputs/outputs. BASIC occupies further channels if the device drivers are addressed with TTL, ADC, REL, MPG IN/OUT. The first IEC-bus instruction and the first graphics instruction also use one channel each so that only the remaining channels are available to the user. If an attempt is made to open more channels than provided by MS-DOS, the error message ERROR 52 "DOS open error" is produced.

Using the CLOSE# instruction, the channel is closed again and the channel number released. The filename is now stored on the floppy disk or hard disk. If an interface has been addressed with the channel number, the interface is deactivated. In the case of the serial interface, for instance, the sync lines are reset. In addition, it should be noted that BASIC provides a buffer with 32 characters for all inputs/outputs to interfaces. In this buffer, data items are first collected and combined by MS-DOS to sectors of 512 characters for the floppy disk or hard disk before they are input or output. If a buffer is only partially filled, it is first output with the CLOSE instruction.

At the end of a program (END instruction or last line), the R&S BASIC closes all open channels.

In the case of the RUN commands, all channels opened in direct mode are closed. This is also the case after an error message in order to make sure that important data items are stored.

If the OPEN instruction includes the following names, BASIC buffers the data in order to transfer them to MS-DOS for a block-by-block transmission:

OPENO#1,"CON:"	for the built-in monitor
OPENI#1,"CON:"	from the installed keyboard
OPENO#1,"PRN:"	for a printer connected to the first Centronics interface
OPENO#1,"LPT1:"	for a printer connected to the first Centronics interface
OPENO#1,"LPT2:"	for a printer connected to the second Centronics interface
OPENI/O#1,"AUX:"	from/for the first serial interface (see manual PCA-B5)
OPENI/O#1,"COM1:"	from/for the first serial interface (see manual PCA-B5)*)
OPENI/O#1,"COM2...6:"	for the other serial interfaces *)
OPENI/O#1,"IEC:"	from/for the IEC-bus interface

Note the colons placed after the interface names. If this indication is missing, the very same interface is addressed all right, but only MS-DOS realizes that an interface and no file is involved. BASIC does not buffer the data which causes the input/output to slow down considerably.

*) not with PSA and PAT

At first sight, it is difficult to understand why an output on the screen requires the whole sequence

```
100 OPENO#3,"CON:2
110 PRINT#3,"MEAN VALUE"
120 PRINT#3,
130 CLOSE#3
```

although the same effect can be obtained by writing

```
100 PRINT "MEAN VALUE"
110 PRINT
```

This may, for example, be due to the fact that inputs/outputs are to be diverted to the keyboard or screen first for test or simulation purposes. The data stream can only be diverted by replacing the string in the OPEN instruction. Alternating output to the screen and the printer is also possible:

```
100 OPENO#3,"CON:":GOSUB Output:CLOSE#3
110 OPENO#3,"LPT2:":GOSUB Output:CLOSE #3

1000 Output:PRINT#3,.....
....
1500 RETURN
```

A further purpose of the OPEN instruction is to set the interface into a particular status. To this end, the string after the colon of the interface name is transferred to the device driver. In the following example, the address 6 is output to the IEC interface and the timer set to 1000 ms for all inputs/ outputs:

```
100 OPENO#3,"IEC:LAD6,TIME 1000"
```

In the case of the serial interface, the data transfer rate and the number of bits per character are determined by the string after COM1, for example in the PCA (see manual PCA-B5).

The syntax is identical with that of the MODE program. At the MS-DOS level, the same IEC-bus setting as above is possible with `MODE IEC:LAD6,TIME 1000`. For setting the other interfaces, refer to the description of the MODE program (see manual "Operating System for PCA" or the manual of the option concerned).

In the examples stated, the PRINT# instruction is always used for the output. Actually, this is the only instruction required for data output. If the data items are to be output to a file and then read in again using INPUT#, note that INPUT#, being an ASCII input, is confined to one line. A CR (carriage return) character must therefore be inserted for the output after the 80th character at the latest. There is no such limitation with the INPUT$ ( ) function, where a character may assume any value from 0 to 255 (8-bit binary value), however the block length must be known.

Therefore, the following example first stores on the file the length of the string which may be up to 30000 characters:

```
100 PRINT#3,LEN(A$)+CHR$(13);
120 PRINT#3,A$;
```

The input instructions first read the number into the variable Count and then the string:

```
200 INPUT#3,Count
230 Strg$=INPUT$(Count,#3)
```

The INPUT$( ) function tries to read the specified number of characters first. If, however, the end of the file is reached before or if the timer becomes active, the number of characters will be smaller than requested or equal to 0, indicating the end.

In the event of the special case number = 1, single characters are read in in order to be checked, e.g., for particular values. If no character is applied, a blank character string (LENG(A$) = 0) is returned.

1.8 The Graphics System

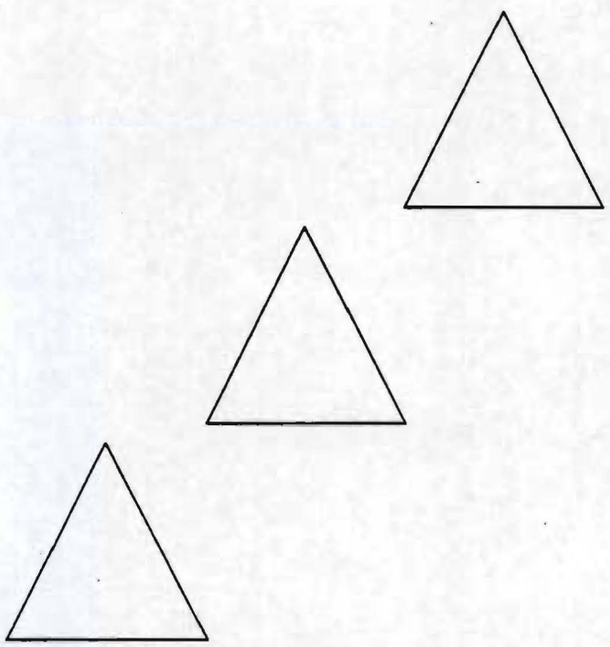
Graphical representations permit to illustrate numbers and other pieces of information very efficiently. Therefore, BASIC offers powerful graphics instructions enabling to draw on the screen, a connected plotter or printer or to transfer the picture to or fetch it from a file.

In order to draw a simple line, first move the imaginary pen to the start point with the desired x- and y-coordinates using the MOVE instruction and then draw the line to the x,y end point using the DRAW instruction.

```
100 MOVE 0,0
110 draw 639,399
```

This example draws a line from the bottom left to the top right of the screen, provided the coordinate system and the display section have not been changed by the WINDOW or VIEWPORT instructions (more about this later).

Using the instruction RDRAW x,y, it is possible to draw relative to the current position of the pen without knowing the absolute x/y-coordinates. Like the parameters of all graphics instructions, these delta X, delta Y values can be numerical expressions, i.e. numerical constants, variables, functions or any combinations produced by arithmetical operations.



```
100 MOVE 100,100: GOSUB Triangle
110 MOVE 200,200: GOSUB Triangle
120 MOVE 300,300: GOSUB Triangle
130 END
140 Triangle:
150 RDRAW 100,0
160 RDRAW  -50,86.6
170 RDRAW -50,-86.6
180 RETURN
```

Similar to relative drawing, the instruction RMOVE x,y permits to move to relative coordinates.

Different "pens" can be used for drawing on the screen. By default, "bright drawing" is set after switching on (SET 1). SET 0 activates the mode "dark drawing" or deleting; bright dots are blanked. SET-1 is used for inverted drawing; if a line is drawn on dark background, this is identical with the default setting SET 1. If, however, the line to be drawn intersects another one, the intersection point is dark (with SET -1). This is an important feature if the line is to be deleted again subsequently. It is simply drawn once again (continue with SET -1); bright dots become dark and the dark intersection point becomes bright again. The intersected line now looks as before, and so do all places that have been overwritten.

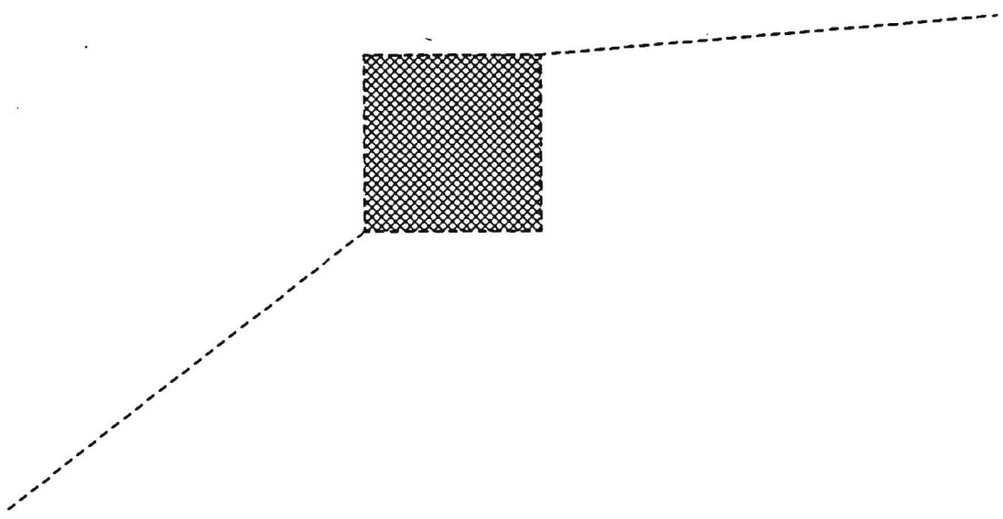
Caution:

Although, basically, the graphics instructions can also be executed on a plotter, this is limited by its physical characteristics. A plotter can only operate in the mode "SET 1". Once selected with SET, the mode ("pen") is retained for all subsequent draw operations until changed by a new SET instruction. (SET includes two further parameters which will be explained at a later date in connection with the color graphics option.)

The next graphics instruction DOT x,y makes a dot at the specified place. As with all graphics instructions, the (imaginary) pen is at the end point when the instruction has been executed, which may then become the start point for the next DRAW instruction.

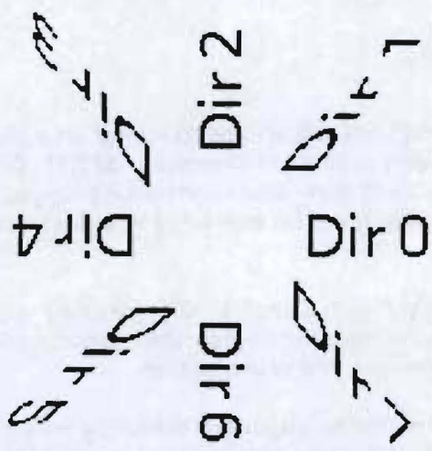
Filled-in rectangles with vertical and horizontal edges are drawn by means of the AREA instruction. One corner is indicated by the position of the pen, the opposite corner is given by the xy-coordinates of the AREA parameters.

The WIDTH instruction determines the fill-in pattern for the rectangles as well as the line type of the DRAW and POLYGON instructions. Each bit of the 16-bit word set to "1" determines that the dot be drawn bright. Zeroes leave the graphics unchanged at the indicated positions. The following example sets a binary string converted with the BIN function for the line pattern, as this is a particularly clear kind of representation. As can be clearly seen in the area drawn with the AREA instruction, the pattern is shifted from line to line.



```
10 WIDTH BIN("1111100000111110")
20 DRAW 200,200
30 AREA 300,300
40 DRAW 600,350
```

For labelling graphics, the LABEL instruction is required. Labelling starts at the top left corner of the first letter. The dot pattern of the characters covers an area of 8 x 8 dots, which, with the size indicated, is multiplied with the second parameter of the instruction. Hence, size 1 covers 16 x 16 dots, size 2 results in 24 x 24 dots and size 3 in 32 x 32 dots. Eight write directions are possible as illustrated in the example below. Line 110 positions the start points of the strings to a circle with the centre point 320, 200 and the radius 50.

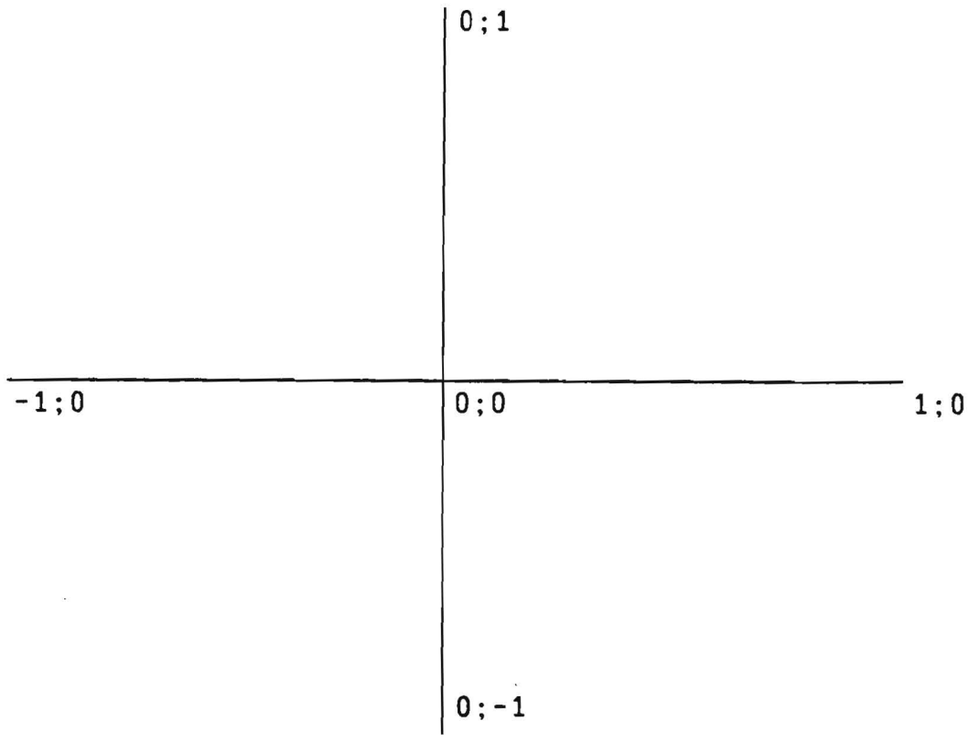



```
100 FOR Dir=0 TO 7
110     MOVE 320+50*COS(PI/4*Dir),200+50*SIN(PI/4*Dir)
120     A$="Dir "+CHR$(Dir+48)
130     LABEL A$,2,Dir
140 NEXT
```

The last draw instruction is POLYLINE. It uses values of an integer field to draw a complete sequence of lines. This is performed at high speed, because no conversions from user into graphics units are performed. For this subject, refer to the following section.

1.8.1 User and Graphics Coordinates WINDOW and VIEWPORT Instructions

The preceding examples assume that the screen or another graphics output device has 640 x 400 pixels and that the user has a coordinate system with the origin in the bottom left corner and the x-axis extending to 639 and the y-axis to 399. Actually, the graphics system is initialized when starting with RUN, however any other (Cartesian) coordinate system can be implemented as well using the WINDOW instruction. The following example shows a coordinate system with four squares extending from -1 to + 1 in the x-direction and from -1 to + 1 in the y-direction, i.e. the origin is in the centre.




```
100 WINDOW -1.2,1.2,-1.2,-1.2,1.2
105 MOVE 0,0: LABEL "0;0",1
110 MOVE 0,0: DRAW 1,0: LABEL "1;0",1
120 MOVE 0,0: DRAW 0,1: LABEL "0;1",1
130 MOVE 0,0: DRAW -1,0: LABEL "-1;0",1
140 MOVE 0,0: DRAW 0,-1: LABEL "0;-1",1
```

Strictly speaking, the coordinate system is specified 20 % larger in the example (up to 1.2) in order for the drawing to remain within the margins.

As the coordinate system covers the entire display area, the x-axis is longer than the y-axis. This distortion can be avoided by means of the VIEWPORT instruction. If the preceding example is extended by

```
90  VIEWPORT 0,399,0,399
```

a coordinate system appears on the screen with equally long axes. 240 pixels on the right-hand side from 399 to 639 are left blank. In order to shift the coordinate system to the centre, the following entry is required:

```
90  VIEWPORT 120,419,0,399
```

The instructions WINDOW and VIEWPORT apply to all following instructions in the program used for moving the pen or drawing (except POLYLINE). The picture built up so far is not changed, however.

1.8.2 Graphics Input/Output

The picture visible on the screen can be transferred to the printer PUD2/3 using the COPYOUT instruction. If 640 pixels are output on the printer in the vertical direction, the distance between the pixels is about 10 % smaller in the x-direction than in the y-direction, which is why the picture appears to be compressed. A maximum of 576 equidistantly spaced pixels is possible (paper format!).Therefore, the user, if he needs a conformal printout, can use the parameter after COPYOUT to indicate which edge(s) of the display is (are) to be omitted.

A picture displayed on the screen can as well be transferred to a file using the instruction

```
100  GSAVE "ABC.PIC".
```

The pixels are read out from the display memory one after the other, the file using about 32 or 38 Kbytes. A color picture uses even four times that much. By entering

```
E> COPY ABC.PIC GRAPH
```

the picture can be transferred back to the graphics device driver and thus be displayed on the screen again at the operating system level. In BASIC GLOAD "ABC.PIC" is used to bring the picture back to the screen where it is superimposed on any display aready present there.

The graphics system can as well use other output devices. After starting, a device driver called GRAPH is loaded as the standard output device for graphics (actually, this is the screen). It is loaded if the file CONFIG.SYS includes the line DEVICE = GRAPHX.SYS. If the DOP is connected and the associated driver loaded with DEVICE = DOPX.SYS, the graphics instructions address the plotter. The output is enabled using the instruction

```
200  GRAPHIC "DOP"
```

In order to address the screen again, use the instruction

```
200 GRAPHIC "GRAPH"
```

Up to four output devices can be indicated, all of them executing the same draw instructions. A filename can also be indicated:

```
100  GRAPHIC "GRAPH", "ABC.MTF"
```

The file ABC.MTF is then assigned all draw instructions of the internal format defined as interface to all graphics units. n. This file can be displayed on the screen using the GLOAD command (GLOAD fetches the point information of GSAVE and the string commands of the GRAPHIC s$ instruction):

```
300  SHELL "COPY ABC.MTF GRAPH"
```

The advantage over the GLOAD, GSAVE instructions lies in the fact that, in particular if only few graphics objects are involved, the file is only a few bytes long and can build up the display considerably faster.

At the operating system level, this file can also be output on the plotter:

```
E> COPY ABC.MTF DOP
```

Parameter /B is required as the file in question is a binary file.

1.8.3 Color Graphics Option PCA-B3 (PCA) or VGA-, EGA Mode (PSA/PAT)

With the color graphics option, each pixel is not only stored in a display memory which contains only the information bright or dark, but in 4 planes. Each pixel can assume $4^2 = 16$ values, i.e. one out of 16 colors at a time. A value is not definitely assigned to a particular color, but accesses a look-up table that can be varied by the user at any time and where a saturation of 0 to 15 can be set for each of the basic colors red, green and blue. Hence, a total of 16 x 16 x 16 = 4096 (PCA) different color shades are availabe (but only 16 at a time). (The data for PSA and PAT can be looked up in the SCREEN instruction described in chapter 2.)

The parameter b of the SET instruction selects the pen to be subsequently used for drawing. The color of this pen which will then be displayed on the screen is only determined by the COLOR instruction (see COLOR).

Pens 1, 2, 4 and 8 draw particularly fast because only one bit is set in each of these binary numbers and only one plane needs therefore be written to. Pen 15 is the slowest. (This does not apply to PSA/PAT;there all pens draw equally fast.)

If two color areas overlap, e.g. the figures drawn with pens 2 and 4, the overlapping area has the same color as pen 6. The area where two colors overlap assumes the color which is produced by logical ORing of the binary numbers of the individual pens.

What has been said so far applies to the non-dominant mode which is set if parameter a of the SET instruction has the values -1 (inverted drawing), 0 (reset dots) or 1 (set dots). If parameter a has the value 2, the dominant mode is cut in. The figure drawn assumes exactly the color which corresponds to the pen; colors drawn first disappear. Thus, all four planes must always be written to which is why the drawing speed decreases exactly as is the case with pen 15 in the non-dominant mode.

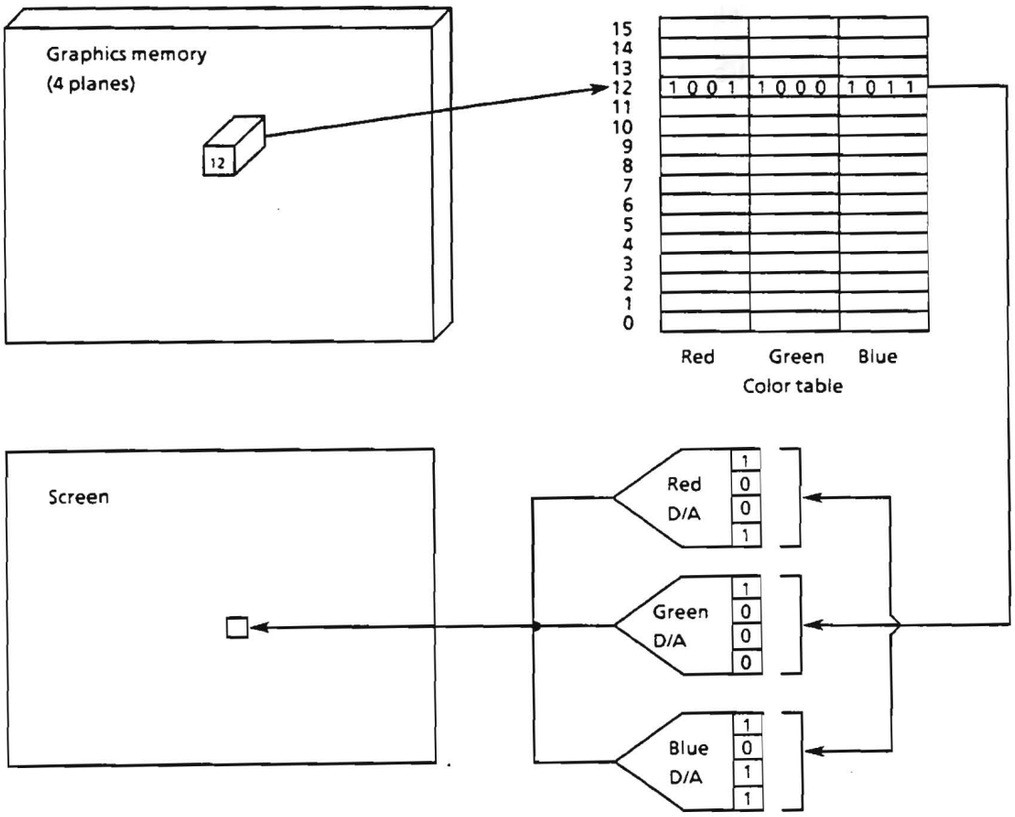


Fig. 4-1 Structure of color graphics option (PCA-B3)

The COLOR instruction determines the colors to be assigned to the individual pens. The number for the pen is followed by the parameter indicating the saturation of the three basic colors red, green and blue. In order to completely fill the look-up table with new values, 16 COLOR instructions are to be written. This is not absolutely necessary, however, as the table is filled with meaningful values following RUN:

The basic colors red, green and blue are assigned to pens 2, 4 and 8 because these pens can draw particularly fast. Without specification of SET, pen 1 is selected, drawing in white in order to remain compatible with programs written for black-and-white graphics.

The mixed colors 6, 10, 12 and 14 are assigned to the colors of the subtractive color palette:

red (2) and green (4) result in yellow (6),
red (2) and blue (8) result in magenta (10),
green (4) and blue (8) result in cyan (12) and
red (2), green (4) and blue (8) result in white (14).
The data for PSA/PAT can be looked up in the COLOR instruction described in chapter 2.

The COLOR instruction also permits to define an additive color palette or any other additive color system. It also applies to what has been drawn before. As the Plotter DOP can only accommodate 8 pens, pen 0 is selected again if 8 is indicated.

A black-and-white printer makes a black dot if any color is set at this position.

The following example illustrates the various possibilities. First, a "circle" subroutine is written using 200 lines to draw a filled-in circle at the position determined by x and y. The values required for this purpose are calculated only once and stored in the field Cir (line 100 to 160).

In the dominant mode, the overlapping sections are completely overwritten by the color of the circle.

If the three overlapping circles are drawn in the non-dominant mode, the mixed colors are produced. Note that the area where all the three colors overlap is white.

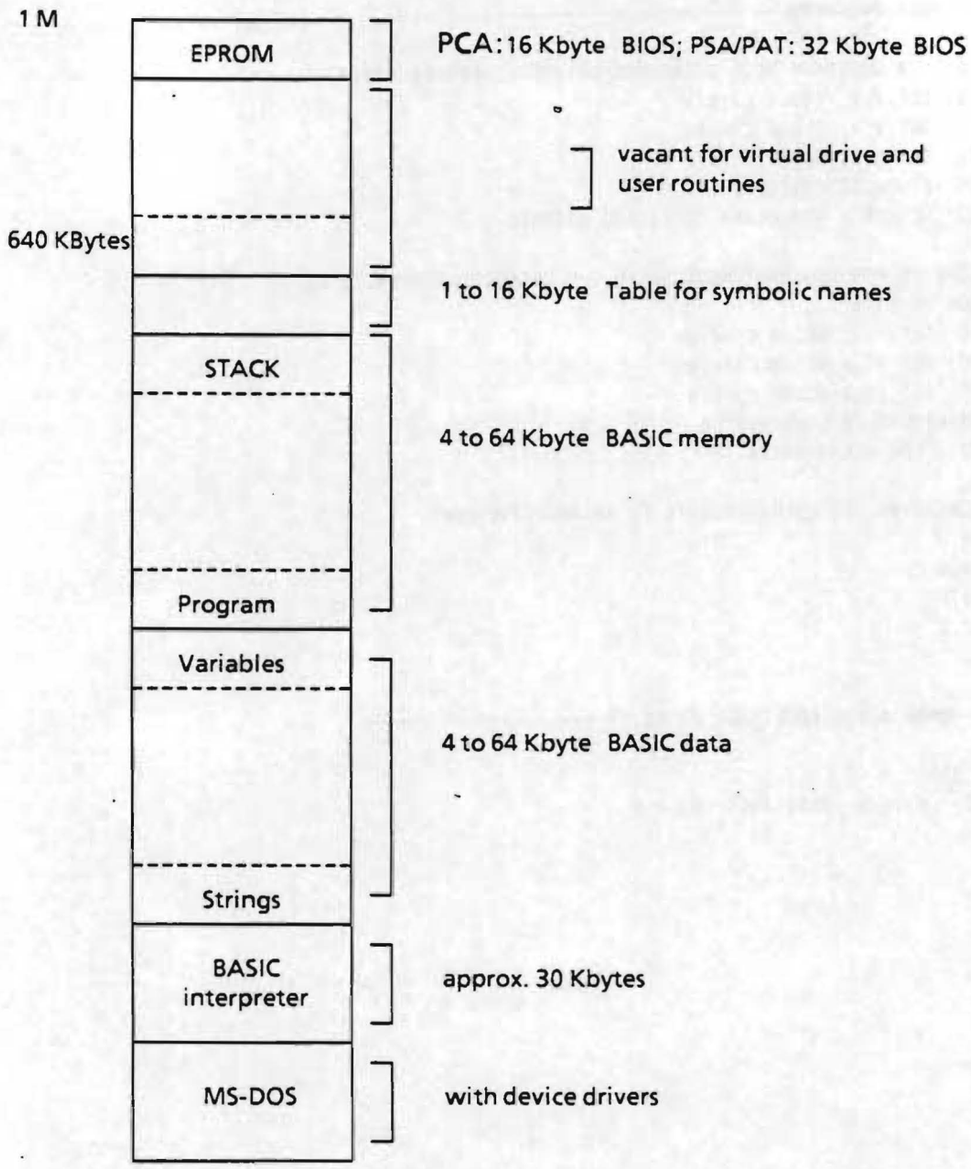
A printer or plotter will show different mixed colors at the overwritten places. This behaviour is simulated on the screen by way of changing to an additive color palette.

```
100 REM ---------FILL THE ARRAY FOR THE CIRCLE ROUTINE----------
110 DIM Cir(200)
120 FOR N=0 TO 200
130     Temp=N-100
140     Cir(N)=SQR(10000-Temp*Temp)
150 NEXT N
160 PRINT "E[2J": REM CLEAR THE ASCII SCREEN
170 REM
180 REM ---------MAIN PROGRAMM-----------------------------------
185 REM
190 PRINT "WRITING IN DOMINANT MODE USING THE DEFAULT COLOR PALETTE"
200 X=150: Y=350: SET 2,2: GOSUB Circle
210 X=250: Y=350: SET 2,4: GOSUB Circle
220 X=200: Y=275: SET 2,8: GOSUB Circle
230 INPUT "PRESS RETURN TO CONTINUE",A
240 PRINT "E[2J"; CLEAR : REM CLEAR ASCII AND GRAPHIC
245 REM
250 PRINT "WRITING IN NON DOMINANT MODE USING THE DEFAULT (SUBTRACTIVE)";
260 PRINT " COLOR PALETTE"
270 X=150: Y=350: SET -1,2:GOSUB Circle
280 X=150: Y=350: SET -1,4:GOSUB Circle
290 X=150: Y=275: SET -1,8:GOSUB Circle
300 INPUT "PRESS RETURN TO CONTINUE",A
310 PRINT "E[2J": REM CLEAR ASCII
315 REM
320 PRINT "NOW CHANGING THE COLOR PALETTE TO AN ADDITIVE ONE"
330 COLOR 6,16,9,0
340 COLOR 10,9,0,9
350 COLOR 12,0,9,5
360 COLOR 14,9,2,2
970 END
980 REM
1000 REM --------DRAW A FILLED CIRCLE AT <X,Y>--------------------
1010Circle:
1020 FOR N=0 TO 200
1030     MOVE X-Cir(N),Y-N: DRAW X+Cir(N),Y-N
1040 NEXT N
1050 RETURN
```

1.9 General Hints for Programmers

1.9.1 Memory Allocation in BASIC

The available memory of 1 Mbyte (PCA) or 640 KBytes(PSA/PAT) is divided up amongst the operating system and BASIC as follows:



1.9.2 Optimum BASIC Speed

The R&S BASIC has been optimized for maximum operating convenience and high execution speed. The computing speed is sufficient for all "normal" control tasks and arithmetic operations. The assembler of the operating system is provided for time-critical parts of program which must operate many times faster than the BASIC interpreter. Thus convenient and fast subroutines can be written in machine language and called by BASIC using CALL.

The execution speed of BASIC programs can be considerably increased by observing the following hints:

- Higher-level instructions are executed more rapidly than if the same function is programmed using individual instructions.

  Example:
  ```
  10 FOR I = 0 TO 100
  20 MOVE I, 0 : DRAW I, 200
  30 NEXT
  ```

  should be replaced by the instruction:

  ```
  10 MOVE 0,0 : AREA 100, 200
  ```

- Time-critical loops should

  a) not contain any instructions which also form part of the program before or after the loop

  b) not contain any REM lines.

- ASCII characters should be printed out as soon as a measured or calculated value is present. The printout of many collected values at once may otherwise fill the input buffer of the printer and thus produce waiting times for the computer.

- Waiting times, e.g. for measured values or response times, can often be used to process and display the previous value. If possible, always commence the new measurement and then evaluate the old one.

- A high data throughput on the IEC bus can also be produced by appropriate programming.

  a) Complex instructions such as IECOUTa,a$ are executed faster than the corresponding individual instructions.

  b) If at all possible, instruments with long measuring times or transients of approx. >100 ms should be programmed such that they signal the presence of a measured value with an SRQ. BASIC can then fetch the value into a subroutine and process other programs during the waiting time (see section 1.5.2).

  c) If possible, the instruments should be programmed such that the result can be used for the controller without further arithmetic or string operations, e.g. the header should be suppressed or the output made in logarithmic form instead of converting linear values in the controller.

  d) Slow or non-standard instruments reduce the data transfer rate on the IEC bus. They should be connected to their own, second IEC bus.

- Data are filed on the hard disk three to ten times as fast as on the floppy drive. In addition, the run-up time during selection is omitted with the hard disk. The writing of data fields in the RAM memory is even faster on the virtual drive which can then be recopied completely onto the floppy or hard disk.

The following section contains a few typical execution times of BASIC functions:

PCA speed	Time per instruction [ms]	
	PCA 2/5	PCA 12/15
A = U	≙ 0	≙ 0
A = RND (V)	0.49	0.32
A = U + V	0.39	0.23
A = V – U	0.42	0.25
A = V * U	0.57	0.33
A = V / U	0.90	0.51
A = SGN (V)	0.20	0.11
A = V AND U	0.45	0.26
A = V OR U	0.43	0.26
A = NOT V	0.25	0.14
A = SIN (U)	6.17	1.92
A = COS (U)	6.19	1.85
A = LOG (U)	5.49	1.58
A = TAN (U)	6.7	1.62
A = ATN (U)	6.2	1.72
A = U ↑ V	1.8 to 15.4	0.4 to 4
A = SQR (V)	2.2	0.53
A = INT (V)	0.36	0.20
A = ABS (V)	0.20	0.11
A = EXP (V)	9.59	3.34
PRINT U;	3.79	3.49
MOVE 0,0: DRAW U,V(U = 50,V = 160)	2.81	
MOVE 0,0: AREA U,V(U = 40,V = 40)	7.81	

The purely numerical calculations are carried out five to ten times as fast with the numeric data co-processor compared to the software emulation. This coprocessor cannot be fitted to the PCA 215; it is, however, fitted to the PCA12/15 as standard and may be fitted to the PSA/PAT as PSAT-B10 option.

1.9.3 Event-controlled Branching

Similar to a real-time operating system, BASIC allows to respond to external events immediately. For this purpose, a running BASIC program is interrupted and continued after execution of the program part related to the event.

The instruction

```
ON <event> GOSUB <line number or label>
ON <event> GOTO  <line number or label>
```

which is to be run through once at the start tells BASIC which program part is to be executed if the event occurs and that it must respond to the event.

Conversely, the instruction

```
ON <event> RETURN
```

prevents BASIC from responding to the event. After running through this instruction, the main program is not interrupted any more; further branching is enabled again by the next ON <event> GOSUB/GOTO INSTRUCTION.

Following branching to the subroutine, renewed calling is inhibited even when the event occurs. (ON <> RETURN is executed internally). This means that the user must explicitely enable renewed branching with ON <> GOSUB/GOTO. This may be done in the subroutine and should then be the last instruction before the RETURN statement separated by a colon.

Example:

```
100 ON SRQ GOSUB 1000
110 REM main program
.
.
.
990 GOTO 110

1000 REM SRQ subroutine
1010 IEC SPL 12,A%
.
.
.
1100 ON SRQ GOSUB 1000: RETURN
```

Branching always takes place at the start of a line to be executed in the program. Line 1100 ensures that branching is enabled and that return to the main program also takes place at any rate. Thus, illegal nesting is avoided.

1.10 BASIC Device Drivers

a) For BASIC it is absolutely necessary to have the display and keyboard driver, named CON and STRIN, containing the screen editor and the program for execution of the ANSI Escape sequences. When this driver has not been loaded several ESC sequences are displayed on the screen which are not executed. The driver becomes available if the device driver STRINX.SYS is loaded with CONFIG.SYS, e.g. by adding the following line to the CONFIG.SYS file

```
DEVICE=\BASDRV\STRINX.SYS
```

Note: ANSI.SYS and the resident programs SIDEKICK and PROKEY do not run, when the BASIC device driver CON/STRIN is loaded.

b) For graphics instructions in BASIC the device driver GRAPH is required. It will become available, if the device driver GRAPHX.SYS is loadad with CONFIG.SYS, e.g. by adding the following line to the CONFIG.SYS file

```
DEVICE=\BASDRV\GRAPHX.SYS
```

Caution: Depending on the hardware the GRAFIC DEVICE DRIVER reserves 24K (CGA), 36K (Hercules), 116K (EGA) or 130K /VGA) memory space for temporary storage of the screen editor and alphanumeric display. It may thus occur that there is not enough memory space for other programs, when the device driver is loaded.

c) For each IEC instruction the IEC bus driver called IEC is required. It will become available by loading the device driver IECX.SYS with CONFIG.SYS, i.e. by adding the following line to the CONFIG.SYS file

```
DEVICE=\BASDRV\IECX.SYS /A:nnn /D:n /I:n
```

The information about the setting of the jumpers and switches on the IEC bus board is provided to the driver by the transfer parameters behind the forward slashes:

/A:nnn indicates the address	2E1 default	Example /A2B8
/D:n indicates the DMA channel	1 default	Example /D3
/I:n indicates the interr. number	7 default	Example/I5

Caution: The IEC bus device driver IECX.SYS disturbs the device driver GPIB.COM of National Instruments, since both drivers access the same hardware. Only the one or the other driver may be loaded with CONFIG.SYS.

d) For printing out the graphics display on a printer compatible to industrial standard (see GSAVE), the printer driver LPT1 is required. It will become available , if the device driver LPTX.SYS is loaded with CONFIG.SYS, e.g. by adding the following line to the CONFIG.SYS file

```
DEVICE=\BASDRV\LPTX.SYS
```

e) The device driver BEEPER is required for generating musical tones using the PLAY instruction. It will become available by loading the device driver BEPX.SYS with CONFIG.SYS, i.e. by adding the following line to the CONFIG.SYS file

```
DEVICE=\BASDRV\BEPX.SYS
```

2 BASIC Instruction Set

2.1 Definitions of Terms Used

In order to keep the description as clear and intelligible as possible, specific letters are uniformly used in the syntax definition.

Character	Meaning	Example
n, m	Line numbers (1 to 65.534)	100
t	String of characters	"Text"
k	Numeric constant	1.234
z	One-digit constant	2
k%	Integer constant (-32768 to 65535)	125
vn	Numeric variable	R1
v%	Integer variable	D1%
v$	String variable	A$
v	One of the variables vn, v%, v$	A, A%, A$
a, b, c, x, y	Constant, variable or numeric expression	1.2 + (A5*4)
s$	String expression (string constant, variable or function)	"VAR" + A$
a/s$	Numeric expression a or string expression s$	—
k/t	Numeric constant k or string constant c	—
( ) < > = # $: " % , ;	Characters belonging to the syntax which must therefore be written.	SIN(a)
[ ]	Brackets enclosing a part of the statement. These are extensions to an instruction which are possible but not absolutely necessary.	
[ ]...	The specified extensions may be repeated in the statement.	

Character	Meaning	Example
Instructions	Components of the program are located immediately after a line number or a colon. If the instruction is written without a line number, it is executed immediately like an instruction (direct mode).	PRINT GOTO
Graphics statements	concern single dot graphics.	
IEC statements	concern the IEC 625bus.	
Commands	Cannot be components of a program since they generally handle programs.	ALOAD NEW
Functions	Always possess an argument. Functions with a subsequent $ character always produce a character string as result (string function). Functions without a subsequent $ character produce a numeric value as result (numeric function).	 CHR$ (10) MID$ (A$) SIN (A)
Synonyms	Notations of an instruction which are accepted as being compatible but are automatically converted internally by the controller into the new syntax.	IECATN IECATT
Default values	Parameters used in BASIC if no parameters are specified in the statement.	
Upper-case letters	mark keywords which must be entered in this sequence. The input may be both in upper case and lower case.	
Lower-case letters	are dummy values for characters or character sequences freely selectable by the user. (The first letter of the examples is written in accordance with the German notation)	
Variable name	is any sequence of letters and digits as well as underline characters used as a means of differentiation. The first character must be a letter. Both upper-case and lower-case letters may be entered: BASIC always converts the first letter to upper case and the subsequent letters to lower case.	A1, A2 Langer_name
Labels	are jump targets of the GOTO/GOSUB instructions, the name establishing the reference. Labels are located after the line number and end with ":". The definition of the variable name given above also applies to labels.	100 Unterroutin_nr1:

2.2 Summary of the BASIC Instruction Set

Numerical functions and operators

> [=]	Greater than (equal to)	
< [=]	Less than (equal to)	Relational operations
< >, =	Not equal to, equal to	
AND		
OR	Boolean operations	
NOT		
XOR		
ABS (a)	Absolute value	
INT (a)	Integer	
SGN (a)	Sign	
SQR (a)	Square root	
SIN (a)		
COS (a)	Angular functions	
TAN (a)		
ATN (a)		
a ↑ b	Power function	
ERL (a)	Poll of error line	
ERM (a)	Error poll	
EXP (a)	Exponent to base e	
FRE (0)	Poll of freely available data storage space	
FRE (1)	Poll of freely available program storage space	
LOG (a)	Logarithm to base e	
RND (a)	Random function	
DEF FN	Functions definable by the user	
FN		

Program execution

BREAK [OFF]	Disable or enabel Break key
BYE oder EXIT	Switch from BASIC to the operating system
CLR	Set basic status
CONT	Program continuation
END	Program end
ERASE v_0 [,v_1]	Clear variables
HOLD a	Waiting time in ms
REM	Remark
RUN [n]	Program start
SHELL [s$]	Invocation of MS-DOS commands
STOP	Program stop
TRACE	Supervision of program run
TRACE a/s$ [,a/s$]...	Program execution in steps with output

Pseudo variables

DATE$	Read out date
DATUM$	Read out date (German)
PI	Circle constant
TIME	Measure time or calculate time
TIME$	Read out time

Data

DATA k_1 [,k_2]...	Setting of data
DIM v (a_1 [,a_2]...)	Array dimensioning
ERASE v	Delete of variables and arrays
INKEY v$	Keyboard poll
INPUT $ (n,[#a])	String input with number of characters
INPUT ["t";] v1 [,v_n]...	Keyboard input
READ v1 [,v_n]...	Reading in data characters
RESTORE [n]	Reset data pointer

Jumps and loops

All GOTOs and GOSUBs may be followed by a line number or a label.

IF a THEN
ELSE
ENDIF
	Structure element (over several lines)

FOR vn = a TO b [STEPc] NEXT [vn]	Loops
IF a THEN...[ELSE]	Conditional branch
GOTO	Unconditional jump
GOSUB	Jump into subroutine
ON COMa GOTO	Enable jump upon end-of-file character in interface
ON COMa GOSUB	Jump into subroutine
ON ERROR GOTO	Enable jump in case of error
ON ERROR GOSUB	Jump into subroutine
ON a GOTO n [,m]... ON a GOSUB n [,m]...	Jump depending on a
ON KEY GOTO ON KEY GOSUB	Enable jump upon Keystroke
ON TIME a GOTO ON TIME a GOSUB	Enable jump at a given time
ON SRQ GOTO n ON SRQ GOSUB	Enable jump upon Service Request
RETURN [a]	Return from subroutine
REPEAT UNTILa	Loop structure (over several lines) Condition at the end
WHILE a WEND	Loop structure (over several lines) Condition at the beginning

Character string processing

ASC (s$	Conversion of ASCII character into numeric value
BIN (s$)	Conversion of binary numbers and decimal numbers
BIN$ (a)	
CHR$ (a)	Conversion of numeric value into ASCII character
HEX (s$)	Conversion of hexadecimal and decimal numbers
HEX$ (a)	
LEN (s$)	Length of a string
LEFT$ (s$,a)	Separate first characters from string
MID$ (s$,a,b)	Remove middle characters from string
+	Linking of character strings
RIGHT$ (s$,a)	Separate last characters from string
STR$ (a[,USING s$])	Conversion of numeric variable into a string
VAL (s$)	Conversion of a string into numeric variable

Edit instructions

AUTO [n] [,Δn]	Automatic line numbering
DELETE n-m	Delete lines
DIR [t]	Output directory
FRE (0)	Available memory space
FRE (1)	
HELP [arg]	Select and display support information texts
IECLIST ON a	Output program on IEC bus
IECLIST OFF	Termination of program output on IEC bus
LIST [n] [-[m]]	Program output on screen
PLIST [n] [-[m]]	Program output on 1st printer interface
NEW	Delete program
RENUMBER [n]	Renumbering of lines
[-m]] [,n] [,Δn]	
SEARCH [n-m,] t	Search text lines
SOFTKEY	Restore softkey labelling
REPLACE [n-m] t_1,t_2	Replace parts of text

Machine instructions

CALL a [,v_1]	Machine program call
CALL# a [v_1]	Call machine routine
INP (a)	Read via addresses
LOAD# a, s$	Load machine routines
OUT a,b	Output via addresses
PASCAL a [,v_1]...	Call Pascal routines
PEEK (a)	Read memory location
POKE a, b	Write into memory location
SEGMENT a oder DEF	Fix a segment
VARPTR (v)	Read in a variable pointer

Input/output via floppy disk, fixed disk and interfaces

ALOAD "t"	Load program stored in ASCII code
ASAVE "t"	Save program in ASCII code
CHAIN s$,m	Reload program sections
CLOSE# [a] [,a]...	Close file
FORM [m-n]	Set page format of printer
INPUT $ (a, [#a])	String input with number of characters
INPUT#a, v_1 [,v_2]...	Load data
LOAD s$ [,R]	Load program
OPENI#a,s$	Open input file
OPENO#a,s$	Open output file
OPENI#a,"CON:	Input/output on console
OPENI#a,"COMb:	Input/output on V24/RS232
OPENI#a,"IEC:	Input/output on IEC bus
OPENI#a,"LPT1:	Input/output on printer
PLAY s$ [,a]	Signal tone with pitch and duration in s$ and repetition rate given by parameter a
PRINT#a, a/s$	Store data
PRINT a/s$	Screen output
PRINT USING	Formatted output
SAVE s$ [,P]	Store program
TAB (a)	Distance form left edge of screen

Graphics instructions

AREA x, y	Draw filled in rectangles
CLEAR	Clear graphic display
COLOR f,r,g,b	Color assignment
COPYOUT [a]	Output graphics to printer
DOT x, y	Draw dot
DRAW x, y	Draw line
GLOAD s$	Load screen from file
GRAPHIC s$,[,s$]	Select graphics interface
GSAVE s$	Store screen in file
INVERT	Invert graphic display
LABEL s$[,a [,b [,c]]]	Labelling of graphics
MOVE x, y	Position cursor
RMOVE x, y	Position cursor relative
POLYLINE a,v%(b)	Draw polyline
RDRAW x, y	Draw line relative
SCREEN a	Fixing screen mode
SET a[,b] [,c]	Display mode for lines and dots
VIEWPORT x1,x2,y1,y2	Fixing display area of screen
WIDTH a	Draw line pattern
WINDOW x1,x2,y1,y2	Fixing coordinate range
ZOOM a	Enlargement and selection of display area

IEC-bus instructions

a) Universal instructions

IEC DCL	Device clear
IEC LLO	Local Lockout
IEC SPE	Serial Poll enable
IEC SPD	Serial Poll disable
IEC PPE k_1 k_2	Parallel Poll enable
IEC PPL v%	Status reply
IEC PPU	Parallel Poll unconfigure
IEC PPD	Parallel Poll disable
IEC TIME a	Set time-out monitor
IEC T1 b	Set time T1
IEC REN	REN line active
IEC NREN	REN line passive
IEC IFC	Transmit IFC
IEC ATN	ATN line active
IEC NATN	ATN line passive
IEC EOI	Output terminator with EOI
IEC NEOI	Output terminator without EOI
ON SRQ GOSUB n	Jump on SRQ
ON SRQ RETURN	Inhibit jump upon SRQ
IEC TERM a	Define input terminator
IEC RLC	Release control
IEC RQS	Send service request

b) Addressed instructions

IEC LAD a	Transmit listener address
IEC TAD a	Transmit talker address
IEC SAD a	Transmit secondary address
IEC UNL	Transmit unlisten
IEC MTA (IEC UNT)	Transmit untalk
IEC SDC	Selected device clear
IEC GTL	Go to Local
IEC GET (IEC GXT)	Group execute trigger
IEC PCON b, k_1,k_2	Parallel Poll configure
IEC PPC	Parallel Poll configure
IEC TCT	Take Control
IEC OUT a_1 [;a_2] ,s$[;]	Transmit character string
IEC $OUT s$	Output character string without address
IEC %OUT a%	Output individual character
IEC IN a_1 [;a_2] ,v$	Read in data
IEC $IN v$	Enter character string without address
IEC %IN v%	Enter character without address
IEC SPL a, v%	Serial Poll
IEC ADR a	Assign address
IEC WMTA	Wait for talker address
IEC WMLA	Wait for listener address
IEC WTCT	Wait for takeover of control
IEC LISTON a	Switch on parallel output on IEC bus
IEC LISTOFF	Switch off parallel output on IEC bus

ABS ABS

Numeric function

Absolute Value Function

Purpose: In order to use the magnitude of a number for further calculation, the sign can be suppressed using the ABS function.

Math.: $x = |x|$

Syntax: `ABS(a)`

a: constant, variable or numeric expression

Example:

```
100 A=ABS(B)
```

This page has been kept free on purpose. The BASIC instruction of an option may be inserted here. The sheets to be inserted are found in the manual of the respective option.

Command

Loading of Programs Stored in ASCII Code

Purpose: This command loads a program in ASCII characters, produced e.g. by means of ASAVE, from floppy disk or fixed disk into the main memory. The syntax is checked at the same time. Program lines present in the main memory are only overwritten if these lines are also used in the program to be loaded. This command cannot be used in a program.

Syntax:

```
ALOAD "program name"
```

Program name, also with drive and search path

Remarks: By use of this command programs can be combined or added to. The line numbers of programs loaded consecutively must, however, match with one another.

Related commands: ASAVE, LOAD, CHAIN

Example: Load the ASCII program TEST.ASC from the default drive

```
ALOAD "TEST.ASC"
```

Load the ASCII program TEST.ASC from the fixed disk drive

```
ALOAD "E:TEST.ASC"
ALOAD "E:\USER\TEST.ASC"
```

Possible error message: Any error messages which would also be produced if this program were entered via the keyboard are also displayed in the status line. The lines with errors are not transferred to the program.

Since ALOAD tests the syntax and pre-compilation takes place, an ALOAD of very long programs may take several minutes.

Instruction

Access to Analog I/O Interface

Purpose: This instruction is used to address the analog interface option PCA-B13.

Syntax:

```
[n] ANG a IN  s$,v
    ANG a OUT s$,b
```

n: line number, also including label
a: number of interface (1 to 3)
s$: string expression for setting the interface
v: numeric variable in which the value is entered
b: numeric expression for the value transferred

Remarks: This instruction is explained in greater detail in the manual of the option. It also contains pages which may be added at a later date in this part of the BASIC manual.

Graphics Instruction

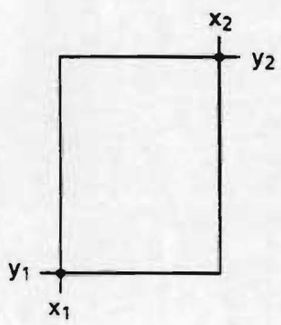
Draw Filled-in Rectangles

Purpose: The AREA instruction is suitable for selecting the graphic background of parts of the screen or, e.g., for drawing tolerance limits for adjustments.

Syntax: `[n] AREA x,y`

n: line number, also including label
x,y: numeric expressions for the x,y coordinates of the target point

Remarks: This instruction draws a rectangular area on the screen. All points within the rectangle are bright, dark or inverted depending on the operating mode selected using the SET instruction. The WIDTH instruction determines the filling-in pattern. The coordinates are determined in a manner similar to the DRAW command, i.e. the starting point is determined by the coordinates x_1 and y_1 of the graphic cursor and the target point is determined by the absolute coordinates of the AREA instruction.



Note that the filling of areas is hardware-based and the CPU time used corresponds to that of the DRAW command.

Related instructions: `MOVE, RMOVE, WIDTH, SET, COLOR`

Example:

```
100 MOVE 200,0
110 AREA 250,50
```

draws a filled-in square

Function

Conversion of Characters into ASCII Values

Purpose: Characters are processed in ASCII code in BASIC, i.e. a value between 0 and 255 is assigned to each character. The numeric value is obtained using the ASC function.

Syntax: `ASC(s$)`
s$: character string or character string constant

Remarks: The numeric value of the first character of the character string is determined in each case (see also ASCII table).

Related Funktion: `CHR$`

Example:

```
100 A=ASC("A")
110 PRINT A
```

Display: *65*

Branch if A$ begins with A

```
200 IEC IN 30,A$:IF ASC(A$)=65 THEN 300
```

Command

Saving Programs in ASCII Code

Purpose: This command is used to save a program in ASCII code on the floppy disk or hard disk so that the program can be processed using editor programs. This command cannot be used in a program.

Syntax:

```
ASAVE [n] [-[m]] "program name"
```

Program name, also with drive and search path

ASAVE n-m	Save from line n to line m
ASAVE n-	Save from line m to end of program
ASAVE -m	Save from beginning of program to line m
ASAVE n	Save line n

Related command: ALOAD, SAVE

Example: Saving the BASIC program as an ASCII file on the default drive

```
ASAVE "TEST.ASC"
```

Saving the BASIC program on the hard disk

```
ASAVE "E:TEST.ASC"
ASAVE "E:\USER\TEST.ASC"
```

Possible error message: The corresponding error messages will be output if the floppy disk is incorrectly inserted or write protected.

Numeric function

Arc Tangent Function

Purpose: This generates the inverted tangent function.

Math.: $y = arc\ tan\ x$

Syntax: `ATN(a)`

a: constant, variable or numeric expression

Related function: `TAN`

Example:

```
100 Y=ATN(X)
```

The inverted function arc sine may be easily calculated from this.

Math.:

$$arc\ sin\ x = arc\ tan\left(\frac{x}{\sqrt{1 - x^2}}\right)$$

```
100 Y=ATN(X/SQR(1-X↑2))
```

The arc cosine function is calculated in a similar manner.

Math.:

$$arc\ cos\ x = arc\ tan\left(\frac{\sqrt{1 - x^2}}{x}\right)$$

```
100 Y=ATN(SQR(1-X↑2)/X)
```

Note: The ATN function presents the result in radians. It is easily convertible into degrees or centesimal degrees by inverting the calculation specified for sine function.

Command

Automatic Line Numbering

Purpose: Line numbers are generated by BASIC using this command; they need not be entered by the programmer. The first line number appears immediately after the command has been entered, each further line number appears when the Return key is pressed. Automatic line numbering is terminated by pressing the Break key.

Syntax: `AUTO [n] [,Δn]`

n: first line number
Δn: increment size

Remarks: Permissible line number range: 1 to 65534
A default value of 10 is inserted for both parameters if no values are specified for n or Δn.

Example: Automatic line numbering from line 100 with a stepwidth of 15

```
AUTO 100, 15
```

Numeric function

Conversion of Binary Number into Decimal Number

Purpose: The BIN function can be used to convert a string of 0 and 1 (binary number) into an integer.

Syntax: `BIN(s$)`

s$: String with 1 to 16 characters containing only 0 and 1.

Related functions: `BIN$, HEX$, HEX`

Example:

```
100 A=BIN ("11111110")
110 PRINT A
```

Display: *254*

Note: The most significant character of a 16-digit number is interpreted as the sign.

String function

Conversion of Decimal Number into Binary Number

Purpose: The BIN$ function can be used to convert an integer into a string (as 16-digit binary number).

Syntax: `BIN$(a)`

a: number between -32 768...32 767 (65 535)

Related functions: `BIN, HEX, HEX$`

Example:

```
100 A$=BIN$(254)
110 PRINT A$
```

Display: *0000000011111110*

Note: With negative numbers, the most significant character is set to 1 and the two's complement is formed.

Instruction

Inhibit or Enable BREAK Key

Purpose: A BASIC program run can be stopped at any time using the BREAK key. It is sometimes necessary to make this key inoperable (e.g. to avoid maloperations).

Syntax: `[n] BREAK Off`

n: line number, also including label

Purpose: The following statement is used to enable the BREAK key function again:

Syntax: `[n] BREAK`

n: line number, also including label

This mode is also switched on by RUN.

Example:

```
.
.
100 BREAK Off
110 REM Protected Program
.
.
.
200 BREAK
210 REM Break-key active
```

Instruction

Return from BASIC to MS-DOS

Purpose: BASIC is terminated using this instruction and the control returned to the operating system.

Syntax: `[n] BYE`

n: line number, also including label

Synonym: `EXIT`

Remarks: BYE may be executed both in direct mode and also under program control. The operating system subsequently signals.

If a BASIC program has been executed in an operating system batch file, the next instruction of the batch file is executed following BYE. In this manner BASIC subroutines can be incorporated into batch files and linked with other programs.

Related instructions: `SHELL, EXIT`

Instruction

Call an External (Machine) Routine

Purpose: This instruction calls an external (machine) routine which starts at address a (offset) (observe segment determination using SEGMENT a). The pointers for the variables are transferred to the routine in the stack. First, the offset is filed in the stack and then the segment. The external routine is terminated by a RETURN instruction (return far/ RETF).

Syntax: `[n] CALL a [,vn1]...`

n: line number, also including label
a: address (offset of routine)
vn_1: numeric variable

Related instructions: `SEGMENT, POKE, CALL#`

Example: Call a subroutine in which only RETF is present

```
10 SEGMENT 4000
20 POKE 1000,HEX("CB")
30 CALL 1000
```

Note: Errors in the called machine routine usually lead to a status which can only be eliminated by RESET (see Section "Incorporation of Assembler Subroutines in BASIC Programs").

Instruction

Call an External (Machine) Routine

Purpose: Call of routine loaded with LOAD#. The number a establishes the relationship between these two instructions. The pointers for the variables are transferred to the routine in the stack. First, the offset is filed in the stack and then the segment. The external routine is terminated by a RETURN instruction (return far/RETF).

Syntax: `[n] CALL# a [,vn1]`

n: line number, also including label
a: link number, mathematical expression with value 1 to 7
vn_1: variable
(See Section 1.4.4 and 1.6)

Related instructions: `LOAD#, CALL`

Instruction

Loading of Program Sections

Purpose: For reloading programs into the main memory under program control and for joining programs together. In contrast to the LOAD instruction, all variables are retained and a line number can be specified up to which the program is to be retained in the memory.

Syntax: `[n] CHAIN s$,m`

n: line number, also including label
m: line number from which the program is to be deleted
s$: program name, possibly with drive and search path

Remarks: The CHAIN instruction may be executed in direct mode from the keyboard and also under program control. The program lines m in the memory are deleted and the specified program is loaded from drive t. The specified program must not contain any line numbers <m.

Under program control it must also be ensured that m is larger than the line number of the called CHAIN instruction since otherwise the BASIC program cannot be continued. The memory space available may become limited since all variables of all subroutines are retained. In this case it is recommendable to delete variables using CLR or ERASE.

Related instructions: `LOAD, ALOAD`

Example: Overlay technique with subroutines

```
100 REM      Main program
.
.
.
500 CHAIN "SUB1.BAS",1000: GOSUB 1000
.
.
.
600 CHAIN "E:SUB2.BAS",1000: GOSUB 1000
.
.
.
700 CHAIN "SUB3.BAS",2000: GOSUB 2000
.
.
.
999 END
1000 REM     Subroutines (SUB1, SUB2)
2000 REM     Subroutine  (SUB3)
```

Possible error message: *ERROR 36: lines nested*

Reason: The first line number in the program to be loaded is smaller than or equal to the last line number of the program present in the memory.

Possible error message: *ERROR 45: CHAIN line erase*

Reason: An attempt was made to delete the line of the calling CHAIN instruction using a parameter m which was too small.

String function

Conversion of Numbers into ASCII Characters

Purpose: The CHR$ function is the inversion of the ASC function. The characters can be determined from the numeric values of the ASCII code.

Syntax: `CHR$(a)`

a: number between 0 and 255

Example:

```
100 A$=CHR$(C*A-5)
```

Related Funktion: ASC

Example: Development of an ASCII table

```
100 FOR I=1 TO 127
110 PRINT CHR$(I),I
120 NEXT
```

Possible error message: *ERROR 40: "parameter too large"*

Reason: The permissible range for the parameter a between 0 and 255 has been exceeded.

Graphics instruction

Clear Graphics

Purpose: This instruction is used to clear the graphics on the screen (PCA). For PSA and PAT this instruction also deletes alphanumeric characters.The parameters set are not reset.

Syntax: `[n] CLEAR`

n: line number, also including label

Instruction

Close a File

Purpose: The instruction CLOSE# serves for the purpose of closing down a file on the floppy disk or fixed disk or terminating a data exchange via an interface.

Each file should be closed as soon as the read or write operation has been carried out in order to prevent the maximum possible number of 15 simultaneously open files from being exceeded and to prevent an open file from being inadvertently opened again in the case of programs with loops or branches.

Syntax: `[n] CLOSE#[a][,a]...`

n: line number, also including label
a: channel number (1 to 15)
(if a is not indicated, all open files are closed)

Related instructions: `OPENI#, OPENO#, INPUT#, PRINT#, INPUT$()`

Example: File 2 is closed:

```
100 CLOSE#2
```

Possible error message: *ERROR 51: "DOS close error"*

Reason: An attempt has been made to close a file although no floppy disk has been inserted.

Possible error message: *ERROR 50 "file not open"*

Reason: An attempt has been made to close a file which has not been opened.

Note: The name of a file will only be stored on the floppy disk or fixed disk if the file is properly closed down with CLOSE#.

Instruction

Setting the Initial State of BASIC

Purpose: The instruction CLR resets BASIC to the state following the RUN command. This may be necessary in the case of long programs which are continuously running or restarting.

Syntax: `[n] CLR`

n: line number, also including label

Remarks: The instruction CLR clears all variables including units and sets them to 0. All FOR-NEXT loops and GOSUB returns are set to the state following the RUN command. Opened files are closed down. RESTORE instruction is issued and the graphics are deleted.

Related instructions: `ERASE, CLEAR`

Instruction

Color Assignment

Purpose:

The color graphics option PCA-B3 (or EGA and VGA mode for PSA/PAT) has 4 memories (planes) for determining the color of each dot. 16 colors can thus be displayed simultaneously. The SET instruction determines the pen to be subsequently used for drawing (see SET). The color of the pen which will then be displayed on the screen is determined by the COLOR instruction. This color also applies to what has been drawn before.

Syntax:

```
[n] COLOR f,r,g,b
```

n: line number, also including label
f: pen 0 to 15 (255*)
r: red portion 0 to 16 (64*) 0: without color
g: green portion 0 to 16 (64*) 16: max. color
b: blue portion 0 to 16 (64*)

*) Look up the values in the SCREEN instruction

Remarks:

After starting the program with RUN, the look-up table has the following values:

Pen	red	green	blue	Color		Remark
				PCA	PSA/PAT	
0 1	0 16	0 16	0 16	black white	black light grey	background
2 3	16 0	0 0	0 6	red dark blue	red light red	
4 5	0 0	16 6	0 0	green dark green	green light green	
6 7	16 6	16 6	0 0	yellow brown	brown yellow	red-green
8 9	0 6	0 0	16 10	blue dark red	blue light blue	
10 11	16 16	0 6	16 0	magenta orange	magenta light magenta	red-blue
12 13	0 4	16 1	16 0	cyan dark brown	cyan light cyan	green-blue
14 15	16 4	16 4	16 4	white grey	grey white	

This table assigns the basic colors red, green and blue to pens 2, 4 and 8, because these pens draw particularly fast.

The mixed colors 6, 10, 12 and 14 correspond to the colors of the subtractive color palette. The COLOR instruction also allows to define an additive color palette or any other color system.

Related instructions: SET, SCREEN, WIDTH

Example: To give an impression of what is possible, the following program draws overlapping filled in circles.

```
5 REM DEFINE VALUES FOR DRAW CIRCLE ROUTINE
10 DIM T(200)
20 FOR N=0 TO 200
30 M=N-100
40 T(N)=SQR(10000-M*M)
50 NEXT N
60 PRINT "Ec[2J": REM CLEAR ASCII SCREEN, CURSOR HOME
70 PRINT "WRITING IN DOMINANT MODE USING THE DEFAULT COLOR PALETTE"
80 X=150: Y=350: SET 2,2
90 GOSUB 320
100 X=250: Y=350: SET 2,4
110 GOSUB 320
120 X=200: Y=275: SET 2,8
130 GOSUB 320
140 INPUT "PRESS <RETURN> TO CONTINUE",A
150 PRINT "Ec[2J": CLEAR
160 PRINT "WRITING IN NON-DOMINANT MODE USING THE DEFAULT (SUBTRACTIVE)";
170 PRINT "COLOR PALETTE"
180 X=150: Y=350: SET 1,2
190 GOSUB 320
200 X=250: Y=350: SET 1,4
210 GOSUB 320
220 X=200: Y=275: SET 1,8
230 GOSUB 320
240 INPUT "PRESS <RETURN> TO CONTINUE",A
250 PRINT "Ec[2J"
260 PRINT "NOW CHANGING THE COLOR PALETTE TO AN ADDITIVE ONE"
270 COLOR 6,16,9,0
280 COLOR 10,9,0,9
290 COLOR 12,0,9,5
300 COLOR 14,9,2,2
310 END
320 REM SUBROUTINE TO DRAW A CIRCLE
330 FOR N=0 TO 200
340 MOVE X-T(N),Y-N: DRAW X+T(N),Y-N
350 NEXT N
360 RETURN
```

Note: This instruction only works with the option PCA-B3 fitted in the controllers PCA2 and PCA12 or, in case of PSA and PAT, with external VGA/EGA compatible color monitor.

Command

Program Continuation

Purpose: If a program has been stopped using STOP instruction or the interrupt key, e.g. in order to read a variable, it can be continued using the CONT instruction.

Syntax: `CONT`

Remarks: The program is continued in the line in which it was interrupted. The program and variable are not affected thereby.

Related instructions: `STOP, TRACE`

Possible error message: *ERROR 43: "can't continue"*

Reason: Program abort with error message or modification of program.

Graphics Instruction

Output of Screen Graphics on the Printer

Purpose: The COPYOUT instruction allows the output of the graphics visible on the screen on the PUD2/3. (Output of the graphics with a graphics printer compatible to industrial standard is effected with GSAVE "LPT1".)

Syntax:

```
[n] COPYOUT [a]
```

n: line number, also including label
a: screen column where printing of the graphics is started (0 to 63)

Remarks: The printout may be made either 1:1 (without parameter) or to scale (with parameter). With a 1:1 printout, the complete graphics display is output on the PUD 2/3. The printout appears to be compressed in the horizontal direction by approx. 10%. If the printout is made to scale, only 576 of the 640 screen columns can be output. Parameter s determines the column where printing of the graphics display is started.

Other areas of the graphics memory, which comprises more than 3 complete pictures, are first made visible on the screen by means of MOVE and ZOOM before they can be printed out.

Related instructions: ZOOM, GSAVE "LPT1"

Example:

```
100 COPYOUT   :REM 1:1 printout not to scale
110 COPYOUT 0 :REM scale printout left-aligned
120 COPYOUT 63:REM scale printout right-aligned
```

Note: ZOOM has no effect on the size of the output. The complete screen is always output.

With the color-graphics option, each colour point set produces a black point on the printout.

Numeric function

Cosine Function

Purpose: This generates the cosine value of the argument in a manner similar to the sine function. The argument is to be indicated in radian measure.

Syntax: `COS(a)`

a: constant, variable or numeric expression

Related Functions: `SIN, TAN, ATN, PI`

Example:

```
100 A=COS(X + 0.5)
```

Instruction

Writing of Data

Purpose: The DATA instruction is suitable for writing data or string constants in the program which are then to be read during program execution by means of the READ instruction.

Syntax: `[n] DATA k1[, k2]`

n: line number, also including label
k: constant or character string constant

Remarks: The data are entered in sequence at any position in the program following the DATA instruction and are separated by commas. They can then be read into the variable in sequence using READ.

If more data are present than space available in one line, they can be written in the subsequent line following another DATA instruction.

The character string constant must be enclosed by quotation marks if it is to contain a comma or a colon.

Related instructions: READ, RESTORE

Example:

```
100 DATA 15.3,20,-17.4
200 DATA "FREQ:",KHZ,LAENGE,0,3M,3MM
.
.
.
300 READ A,B,C,A$
320 READ B$
400 PRINT A$;A;B$
```

Output: *FREQ: 15.3KHZ*

Pseudo variable

Readout of Date

Purpose: The DATE$ variable is used to read the date into a character string variable.

Syntax: `DATE$`

Remarks: The assignment of DATE$ to a character string variable results in a string with 8 characters of the form: mm-dd-yy.

mm = month
dd = day
yy = year

The date is set at the operating system level using the instruction DATE.

If the option PCA-B10 (real-time clock) is fitted, the date is obtained therefrom.

The year is set in the operating system using MODE:CLOCK:year.

Related variables: `DATUM$, TIME, TIME$`

Example:

```
100 A$=DATE$
```

Pseudo variable

Reading of Date
(German Notation)

Purpose: The DATUM$ variable is used to read the date in German notation into a string variable.

Syntax: `DATUM$`

Remarks: The assignment of DATUM$ into a string variable results in a string with 8 characters in the form:

```
tt-mm-jj.
```

tt = day
mm = month
jj = year

The date is available at the operating system level using the DATE instruction.

The date is obtained from the option PCA-B10 (real-time clock) if present.

The year is set in the operating system using MODE CLOCK:year.

Related variables: `DATE$, TIME, TIME$`

Example:

```
100 A$=DATUM$
```

Numeric function (definition)

User-definable Function

Purpose: To enable user to define a function. This is particularly useful if the formula used is very long or if it is frequently called up in the program.

Syntax:

```
[n] DEF FN t(vn)= a
```

n: line number, also including label
t: function name
vn: numeric variable upon which the function depends
a: mathematical expression which may contain vn

The expression a usually also contains the transferred variable vn. This variable is assigned its original value after being called, but expression a is calculated with the value transferred as parameter vn.

Remarks: Several functions with different names can be defined in the program.

The function with the same name can be defined several times and the last definition before calling is valid.

Related function: FN

Example:

```
100 DEF FN T(X)=14*X↑2+2*X
200 Y = FN T(A)
```

Command

Deletion of Program Lines

Purpose: The DELETE command is suitable for deleting existing lines or sections in the program.

Syntax: `DELETE [n] [-[m]]`

DELETE n-m delete from line n to line m
DELETE n- delete from line n to end of program
DELETE -m delete from start of program to line m
DELETE n delete line n

Remarks: Individual lines may also be deleted by entering the line number and pressing the Return key (enter an empty line).

Related instructions: NEW

Example: Delete program lines 100 to 200

```
DELETE 100-200
```

Note: If a line number specified in the DELETE command does not exist, the range between the indicated line numbers will be deleted. If the specified line numbers exist, they will be deleted as well.

Instruction

Array Dimensioning

Purpose: An array must first be dimensioned with respect to type and size before it can be used. The DIM statement is used to this end.

Syntax: `[n] DIM v(a1 [,a2]...)`

n: line number, also including label
v: numeric or character string variable, also further variables, separated by comma
a_1: highest index which limits the field size (also variable or expression)

Remarks: The controller then reserves $a_1 + 1$ or $(a_1 + 1)$ $(a_2 + 1)$... memory locations for the dimensioned array. The array elements are simultaneously filled with 0 in the case of numeric variables and with the empty word in the case of character strings.

Related instruction: ERASE

Example:

```
100 DIM A(28)
200 DIM Cx%(14)
300 DIM A1$(F)
350 DIM Integer_field%(Dim_1), String_field(Dim_2)
```

multi-dimensional arrays:

```
400 DIM A$(I,14)
500 DIM B(K+12,I2,4/A%)
600 DIM Field(10,10,10), measured values (number, rows)
```

Possible error message: *ERROR 20: "redimensioned array"*

Reason: Array has already been dimensioned. This prevents a data array from being deleted by new dimensioning.

Note: Several arrays to be dimensioned are separated by commas in the DIM instruction.

Command

Output of Directory

Purpose: Similar to the DIR command of the operating system, the DIR command in BASIC outputs the directory of the floppy disk or fixed disk onto the screen.

Syntax: `DIR [filename]`

Filename, also with drive and search path

Remarks: Just as under the operating system, filenames and pathnames can be specified. The jokers ? and * in the BASIC DIR command replace characters or character groups. Five filenames per line are output on the screen.

Related instructions: SHELL

Example:

```
DIR
DIR *.BAS
DIR A:\USER\ *.*
```

Graphics instruction

Drawing of Dots

Syntax:

```
[n] DOT x,y
```

n: line number, also including label
x,y: numeric expressions for the x/y coordinates

Remarks:

The instruction draws a dot whose location is determined by the coordinate x and y.

Related instructions:

MOVE, RMOVE, SET, DRAW, RDRAW

Example:

```
100 FOR A=0 TO 2*PI STEP .0314
110 X=95*COS(A)
120 Y=95*SIN(A)
130 DOT 160+X, 100+Y
140 NEXT
```

A circle comprising dots is drawn using the following example. The centre point has the coordinates x = 160 and y = 100. The radius corresponds to the distance of 95 dots.

Graphics-Instruction

Drawing of Lines

Syntax: `[n] DRAW x,y`

n: line number, also with label
x,y: numeric expressions for the x/y coordinates of the target point

Remarks: This instruction draws a visible line on the screen which ends at the point of intersection of the x and y coordinates.

The starting point of the line must be determined before this instruction is used for the first time. This is carried out using the instruction MOVE or DOT. Further DRAW instructions can then follow in the program. Lines are then joined together. Each new DRAW instruction draws a new line which ends at the x and y values of this instruction and restarts where the line of the previous instruction has stopped.

Related instructions: `RDRAW, MOVE, RMOVE, POLYLINE, SET, WIDTH`

Example: Two crossed lines are drawn on the screen using the following example.

```
100 MOVE 110,50
110 DRAW 210,150
120 MOVE 210,50
130 DRAW 110,150
```

Note: Drawing outside the screen coordinates leads to the overwriting of the video memory.

Instruction (structure element)

Marking the ELSE Branch of the IF Instruction

Purpose: Marks the beginning of the ELSE branch from an IF-THEN/end of line instruction to ENDIF line block.

See IF instruction (syntax 3:)
See ENDIF instruction

Syntax: `[n] ELSE`

n: line number, also including label

Related instructions: `IF THEN, ENDIF`

Note: The keyword ELSE may as well be part of an IF line and will then follow the keyword THEN.

See IF instruction (syntax 2:)

Possible error message: *Error 48: "no IF-struc., [ELSE], ENDIF match"*

Reason: No IF-THEN/end of line instruction active of ENDIF not suitable.

Instruction

Program End

Purpose: The END instruction may but need not be at the end of a program. It causes the program to halt and stops the controller from continuing with the subsequent subroutines at the end of the main program. Thus it avoids the occurrence of error messages.

Syntax: `[n] END`

n: line number, also including label

Remarks: The program is stopped with the END statement without changing the variables and the controller signals READY.

The END instruction may be used several times in a program which may be useful e.g. with program branching.

Related instruction: STOP

Example:

```
1000 END
```

Instruction (structure element)

Purpose: Marks the end of the line block begun with the IF-THEN/end of line instruction.

See IF instruction (syntax 3:)
See ELSE instruction

Syntax: `n ENDIF`

n: line number, also including label

Related instructions: `IF THEN, ELSE`

Possible error message: *Error 48: "no IF-struc., [ELSE], ENDIF match"*

Reason: No IF-THEN/end of line instruction active.

Instruction

Clearing Variables

Purpose: The ERASE instruction is used to clear variables, especially in dimensioned arrays.

Syntax: `[n] ERASE v0 [,v1]...`

n: line number, also including label
vi: variable to be cleared

Remarks: ERASE clears the variable or the variable field with the name v. The value in brackets serves as a dummy value. The memory location occupied by the variables will then be available again for use as required. It is, however, not available for strings. Thus, deleting a string array creates storage space for the pointers. The space occupied by the string is, however, not available again.

Related Function: `DIM, CLR`

Example:

```
100 ERASE A(0),B$,A%(0)
```

Function

Reading of Error Line

Purpose: This function may be used to determine the number of a faulty line. This is useful e.g. for debugging in a subroutine called using ON ERROR GOSUB.

Syntax: `ERL(a)`

a: dummy value, any value

Related function: `ERM, ON ERROR`

Function

Reading the Error Number

Purpose: This function checks the error status of the controller. This is useful for example for the evaluation of an error in a subroutine called up by ON ERROR GOSUB.

Syntax: `ERM(a)`

a: Dummy value, any value

Remarks: The error status is normally 0. ERM(a) assigns a value between 1 and 76 if an error occurs. The associated error messages are described in Section 3.

Related functions: `ERL, ON ERROR`

Example: Detecting a time-out on the IEC bus:

```
100 ON ERROR GOSUB 1000
110 IECOUT 28,"123"
.
.
.
1000 A=ERM (0)
1010 IF A=10 THEN PRINT "TIME OUT"
1020 ON ERROR GOSUB 1000
1030 RETURN
```

Instruction

Return from BASIC to MS-DOS

Purpose: BASIC is terminated using this instruction and the control returned to the operating system.

Syntax: `[n] EXIT`

n: line number, also including label

Synonym: `BYE`

Remarks: EXIT may be executed both in direct mode and also under program control. The operating system subsequently signals.

If a BASIC program has been executed in an operating system batch file, the next instruction of the batch file is executed following EXIT. In this manner, BASIC subroutines are incorporated into batch files and linked with other programs.

Related instructions: `SHELL, BYE`

Numeric Function

Exponential Function

Purpose: The exponential function works with base e.

Math.: $y = e^x$

Syntax: `EXP(a)`

a: constant, variable or numeric expression

Related function: `LOG`

Example:

```
100 Y=EXP(X)
200 PRINT EXP(2)
```

Result: *7.389056098931*

Possible error message: *ERROR 31: "numeric overflow"*

Reason: Exponent too large

Numeric function (call)

User-definable Function

Purpose: The function, once defined, may be called as often as required at different positions in the program either directly or by the program and is still in store at the end of the program.

Syntax: `FN vn(a)`

vn: function name
a: argument whose function is to be calculated

Remarks: It must be ensured that the first call in the program is not present before the definition. Variables used for the calculation retain their value assigned to them.

Related functions: `DEF FN`

Example: The hyperbolic sine function

Math:

$$y = \frac{e^{x} - e^{-x}}{2}$$

```
100 DEF FN Sh(X)=(EXP(X)-EXP(-X))/2
200 Y=FN Sh(.8)
```

The function is first defined and then called, where y is assigned the sine hyperbolic 0.8.

Possible error message: *ERROR 47: "undefined Function"*

Reason: Called function has not yet been defined.

Instruction

Loops

Purpose: In many applications, a program loop is executed with different values of a variable.

Syntax:

```
n FOR vn = a TO b [STEP c]
m NEXT [vn] [,vn]...
```

n: first line in the loop, also including label
m: last line in the loop, also including label
vn: floating point variable
a: initial value (constant, variable or numeric expression)
b: final value (constant, variable or numeric expression)
c: increment size (constant, variable or numeric expression). The increment will be set to 1 if STEP is not specified.

Remarks: The controller sets the loop variable vn equal to a when the execution of the instruction starts. The loop variable is increased by the increment c each time the NEXT statement is performed and the loop between n and m executed again. This process is repeated until the variable vn becomes larger than the final value b and the program located after m is then processed. The loop is executed once in any case. After termination of the loop, the variable vn has the value b + c.

The loop variable need not be specified in the NEXT instruction. Contrary to the general definition of variable names, the loop variable vn may contain up to 16 characters (letters, digits, underline characters). FOR-NEXT loops may be nested max. 22-fold and contain other structure elements as well.

Example:

```
100 FOR A=1 TO 10
120 NEXT
```

Command

Producing Page Format for Printer

Purpose: An upper and lower margin is produced for all outputs on the printer PUD. A form feed is also executed prior to all printer outputs (PLIST, COPYOUT und OPENO#a,"LPTb:")

Syntax: `FORM [m - n]`

m: first line to be printed
n: last line to be printed
(max. 66 in the case of 11" paper)
(max. 72 in the case of 12" paper)

The parameters are omitted in order to switch off this function.

Syntax: `FORM`

This function is switched off after the BASIC interpreter has been loaded.

Related commands: `PLIST, COPYOUT, OPENO LPT`

Example:
```
FORM 12-61
```

Numeric function

Freely Available Memory Locations

Purpose:

Each character and each instruction of a program requires a memory location.

The longer a BASIC program, the smaller the originally available memory location.

If information on the available memory locations is required during programming, the function FRE(1) is used.

Data used during execution of the program, i.e. strings and numerical values, also require memory locations. These are filed in their own memory segment.

The memory locations available for data are read out using FRE(0).

Syntax:

```
FRE (a)
```

a = 1: free program memory
a = 0: free data memory

Example:

Output the available memory space for the program:

```
PRINT FRE(1)
```

Output the available memory space for the data:

```
PRINT FRE(0)
```

Remarks:

If possible 64 k are reserved for the program and the data when the BASIC interpreter is loaded. If the available memory is smaller it will be divided up at a ratio of 4:4:1 for labels. The memory polled with FRE ( ) is slightly smaller because part of the memory is internally used by BASIC.

Instruction

Load Graphics from File

Purpose: Graphics drawn on the screen can be stored in a file on the floppy disk or hard disk using the GSAVE instruction. With the GRAPHIC instruction the information about what is to be drawn can be stored in a file. GLOAD is used to bring both formats back to the screen again.

Syntax: `[n] GLOAD s$`

n: line number, also including label
s$: filename, also with drive and search path

Related instructions: `GSAVE, LOAD, GRAPHIC`

Example:
```
100 GLOAD "A:\USER\TEST.BLD"
```

Note: If there is already a graphics display on the screen, both pictures will be superimposed following GLOAD. In order to prevent this, the CLEAR instruction may be used to clear the screen prior to loading.

The graphics is always loaded into the part of the graphics memory currently visible on the screen. Note that a further part of the graphics memory can also be made visible using ZOOM.

Instruction

Branch to a Subroutine

Purpose:

In order to arrange BASIC programs as clearly as possible, frequently used sections of larger programs should be used as subroutines and inserted at the required positions in the program. This arrangement saves both memory space and program writing.

Syntax:

```
[n] GOSUB m
[n] GOSUB marker
```

n: line number, also including label
m: 1st line of the subroutine label
label: any sequence consisting of letters, digits and underline characters

Remarks:

The program jumps to the first line of the subroutine with the GOSUB statement and returns to the next instruction after the GOSUB statement once it has been processed (see RETURN). Another subroutine can be called within the first subroutine. Up to 22 subroutines may be nested in this manner.

If GOSUB is not followed by a line number but by a character sequence starting with a letter, BASIC searches for the line of this label. Note that this label directly follows a line number and must end with a colon.

Related instructions:

RETURN, GOTO, ON GOSUB

Example:

```
100 GOSUB1000
1000 REM SUBROUTINE
1010 RETURN
```

or

```
100 GOSUB SUBROUTINE
1000 SUBROUTINE: PRINT A
.
.
.
1100 RETURN
```

Instruction

Jump to a New Line Number or Label

Purpose: The GOTO statement is used to jump to a new line number in the program and to continue execution from there.

Syntax:

```
[n] GOTO m
[n] GOTO mark
```

n: line number, also including label
m: line number to be jumped to label
label: any character sequence consisting of letters, digits and underline characters

Related instructions: GOSUB, ON GOTO, IF

Example:

```
100 GOTO300
200 REM SKIPPED PROGRAM
300 REM JUMP ADDRESS
```

or

```
100 GOTO Not_set
200 REM
300 Not_set:
```

possible error message: *ERROR 25: "undefined line"*

Reason: The line jumped to does not exist.

Note: Direct entry of the GOTO statement is used to continue the program at another position without deleting the variables as in the case of RUN.

The program must not be modified, e.g. by entering new lines. In this case, undefined errors are produced after GOTO n.

Instruction

Graphics Output on Interfaces or Files

Purpose: The graphics instructions permit to draw not only on the screen but for example also on the connected plotter DOP. Besides, the information to be drawn can be stored in files. GLOAD brings the drawing back to the screen again.

Syntax:

```
[n] GRAPHIC s$ [, sn$]...
```

n: line number, also including label
s$: ont to a max. of 4 character strings of the names of the interfaces or files, also with drive and search path

Related instructions: GLOAD,GSAVE, COPYOUT

Example:

`GRAPHIC "GRAPH"`	Output on screen; this is initialised by RUN
`100 GRAPHIC "GRAPH", "DOP"`	Graphics are drawn both on the screen and the plotter at the same time
`200 GRAPHIC "DAT.MTF"`	A file of graphics objects is produced

Note: The file or interface is opened using this procedure. Data are transferred using the following graphics instructions. The file is not closed, i.e. finally recorded, until the END instruction or the maximum line number have been reached.

If another GRAPHIC instruction follows, the open files will be closed as well, also in the case of RUN, because some files might still be open due to an interruption in the previous program run.

If a file is specified as output unit, it will be filled with data which may then be further copied by MS-DOS using the COPY command. The file DAT.HTF produced in the example above can be output on the screen by means of

```
COPY/B DAT.MTF GRAPH
```

or transferred via the loaded device driver DOPX.SYS to the connected plotter DOP by means of

```
COPY/B DAT.MTF DOP.
```

The parameter /B is required because the file has binary values.

Instruction

Saving Screen Graphics in a File or Output on a Printer

Purpose:

A graphics drawn on the screen can be stored point by point in a file on the floppy disk or hard disk using this instruction. GLOAD is used to bring it back to the screen again.

If the printer interface LPT1 (and PRN, resp.) or LPT2 are indicated as output file, the point information will be transmitted to a connected IBM Graphics printer (or compatible printer) and printed.

For the printer of type PUD2/3 the COPYOUT instruction is used to this end.

Syntax:

```
[n] GSAVE s$
```

n: line number, also including label
s$: filename, also with drive and search path

Example:

```
100 GSAVE "A:\USER\TEST.BLD"
```

Related instructions:

GLOAD, COPYOUT, ZOOM, GRAPHIC

Remarks:

The part of the graphics memory currently visible on the screen is always saved. Note that a further part of the graphics memory can also be made visible using ZOOM.

Command

Call of HELP Program

Purpose: The HELP program displays support information texts, which similar to the BASIC manual further specify all instructions, functions and commands, on the screen.

The contents output on the screen prior to calling of the HELP program is regained when the HELP program is left again.

Syntax: *HELP[arg$_1$ [arg$_2$ ... [arg$_n$]]...]*

Remarks: Following calling of the HELP program without indications of arguments, the support information required is provided for the user via help menues.

If arguments are supplied when starting the program, the support information selected is immediately indicated. The arguments may be abbreviated as long as they prove umambiguous.

Related command: SHELL

Example:

```
HELP

HELP PRINT      or
HELP PR
```

Note: The HELP program is installed on the hard disk according to data file HELP.DOC with installation program HINST.BAT containing auxiliary text information in German or English. The programs are on the system floppy disk.

Numeric function

Conversion of Hexadecimal Numbers into Decimal Numbers

Purpose: A hexadecimal number can be processed as a decimal number using the function HEX.

Syntax: `HEX(s$)`

s$: hexadecimal character string consisting of 0 to 9 or A to F in the range between 0 and FFFF

Related functions: `HEX$, BIN, BIN$`

Example:

```
100 PRINT HEX("2F3A")
```

Display: `12090`

Note: It should be noted that hexadecimal numbers greater than 7FFF exceed the limit of 32767 according to the definition of integer variables so that the next higher value begins again at the bottom end of the range, i.e. at -32767. FFFF then corresponds to the decimal value -1, i.e. a decimal value of 65536 must always be added when converting hexadecimal numbers larger than 7FFF.

String function

Conversion of Decimal Numbers into Hexadecimal Numbers

Purpose: A decimal number can be converted into a hexadecimal number using the function HEX$.

Syntax: `HEX$(a)`

a: number -32768 to 32767 (65535)

Related functions: `HEX, BIN$, BIN`

Example:

```
100 PRINT HEX$(64000)
```

Display: `FA00`

```
110 A$=HEX$(Numvar)
```

Instruction

Waiting Time

Purpose: The HOLD statement can be used to stop a program for a specific period.

Syntax:

```
[n] HOLD a
```

n: line number, also including label
a: waiting time in ms

Remarks: The waiting time may be a constant, a variable or an arithmetic expression but must be in the range from -32768 to 65535.

Example:

```
100 HOLD C*300
```

Note:. The waiting time is realized by a software loop. Tolerances are in the range of approx. ± 20%. A more accurate waiting time can be realized by working with the TIME-Pseudo variable.

Related function: TIME

IEC instruction

Assignment of an IEC-bus Address

Purpose: The controller usually has the system control and does therefore not require its own IEC-bus address. However, if several controllers are connected to the bus, each must have an address just like IEC bus devices. The controller can then respond when its address is called. The IECADR instruction is used to assign this address.

Syntax: `[n] IEC[z]ADR b`

n: line number, also including label
z: number of the IEC bus 1 or 2, default value 1
b: IEC address 0 to 31, initialized 31

Remarks: The IECADR instruction has a function similar to setting of the IEC address on devices using a DIP switch. The assignment is made simultaneously for the talker and listener address. The address may be changed during program execution.

Example: Reading in of IECOUT10,"DATA"

```
90 IEC TERM 13: IEC TIME 1000
100 IEC RLC
110 IEC ADR 10
120 IEC $IN A$:PRINT A$
```

Display: *DATA*

IEC instructions

Line Messages

Purpose: The controller provides the facility for also controlling the management lines via BASIC instructions (except the line "SRQ" which is not controlled by the controller). However, this will be necessary only with special programming functions, however, since the controller automatically controls the management bus with all IEC instructions as prescribed in the standard.

Line ATN informs all the connected interfaces that the information contained on the data bus has to be considered (IEC Command).

Syntax: Line message

`[n] IEC[z]ATN` Line "ATN" active.

`[n] IEC[z]NATN` Line "ATN" passive.

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Synonym:
`IEC[a]ATT`
`IEC[a]NATT`

Possible error message: *not an IEC-bus controller*

Reason: IECATN may only be transmitted if the computer is the controller (not talker/listener).

IEC instruction

Device Clear

Purpose: The instruction sets all devices into a basic status (defined by the manufacturer) and should be used before each new use of the bus and at the start of a program.

Syntax: `[n] IEC[z]DCL`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Remarks: The instruction outputs the control character 20 (decimal) together with ATN.

Example:

```
100 IEC DCL
```

IEC EOI
IEC NEOI

IEC instruction

Definition of the Output Delimiter with/without EOI

Purpose: This instruction can be used during the output of data to select whether the last character in a string is transmitted with or without the line message EOI.

Syntax:

```
[n]IEC[z]EOI
[n]IEC[z]NEOI
```

n: line number, also including label
z: number of the IEC bus 1 or 2, default 1
Default: with EOI

Remarks: The last character of each OUT instruction is transmitted with EOI (low = true) following entry of IEC EOI or the program start. If the OUT statement contains only one byte (e.g. IEC %OUT), this is transmitted together with EOI. If one or more delimiters follow the character string (e.g. IEC OUT 10,A$), only the last delimiter is transmitted together with EOI.

The EOI line is not serviced following the IEC NEOI statement and remains inactive until reactivated with IEC EOI.

Example:

```
100 IEC EOI
110 IEC OUT 4,"FE 14":REM LF with EOI
120 IEC OUT 4,"CS 32";:REM 2 with EOI
.
.
.
200 IEC NEOI
220 IEC OUT 4,"AC 36":REM without EOI
```

IEC instruction

Group Execute Trigger

Purpose: Operation is triggered in the addressed device or in a group of devices by the IEC GET statement. This is important for time-critical sequences.

If, for example, the charging curve of a capacitor is to be recorded, the voltage source, the voltmeter and a plotter can be started simultaneously using IEC GET.

Syntax: `[n] IEC[z]GET`

n: line number, also including label
z: number of IEC bus 1 or 2, default 2

Synonym: `IEC GXT`

Remarks: The statement outputs the control character 8 (decimal) together with ATN. The devices to be triggered must first be addressed as listeners before executing GET.

Example:

```
100 IEC LAD 4,LAD 7,LAD 11
110 IEC GET
```

IEC instruction

Go to Local

Syntax: `[n] IEC[z]GTL`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Remarks: The addressed devices can be manually serviced again following this statement. The effect of this instruction is terminated by addressing the device again as a listener. The lockout of the remote control (see IEC LLO) is also disabled at the same time.

The statement outputs control character 1 together with ATN.

IEC instruction

Line Messages

Purpose: Only the system controller may activate line IFC and thus reset all the interfaces connected to the bus into their basic status.

Syntax: Line message

`[n] IEC[z]IFC` IFC line < 100 µs (about 1 m) active

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Remarks: IECIFC may be transmitted only if the instrument is the controller.

Possible error: IFC does not go low (true).

Possible reason: Jumper not plugged in on system controller (see also Section on IEC bus operation preparations)

IEC instruction

Input of Strings with Addressing

Syntax: `[n] IEC[z]IN a1 [; a2], v$`

n : line number, also including label
z : number of IEC bus 1 or 2, default 1
a_1: talker address (0 to 31) *)
a_2: secondary address (0 to 31) *)
v$: character string variable into which the data is read.

Remarks: The individual steps of the instruction are:

a) Addressing of device with talker address a. (TADa)
b) A secondary address (a) is transmitted if specified. (SADa)
c) Transmission of own listener address. (UNL)
d) Reading-in of data in "v$" up to the delimiter according to IECTERM.
e) Deaddressing of talker (MTA).

Example:

```
100 IEC TERM 13
110 IEC TIME 20
120 V=6
.
.
.
300 IEC IN V,U$
330 PRINT U$
```

100: Delimiter "CR" for end of data.
110: 20 ms time-out for handshake.
300: Data are read into U$ by the talker with address 6.
330: The read-in data are output on the screen.

*) Decimal value of bits b_1 to b_5

IEC $IN
IEC %IN

IEC instruction

Input of String Variables or Integers without Addressing

As controller and talker/listener:

Syntax: `[n] IEC[z]$IN, v$`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1
v$: string variable to be read into

Input of a single character:

Syntax: `[n] IEC[z]%IN, v%`

n: line number, also including label
z: number of IEC bus 1 or 2, default
v%: fixed-point variable to be read into

Remarks: The computer can only act as a listener in the two instructions. The characters are received if a talker has been addressed before or a device operates in 'talk only' mode.

When the computer has released control it will as talker/listener receive the information with these instructions.

Example: Read-in of a byte from the device with address 6:

```
100 IEC TERM 13
110 IEC TIME 20
120 V=6
.
.
.
300 IEC TAD V
310 IEC %IN U% 320 IEC MTA
330 PRINT U%
```

Read-in of all data up to delimiter CR from a 'talk only' device:

```
100 IEC TERM 13
120 IEC $IN A$
140 PRINT A$
```

Note: When a running program is aborted with BREAK while a talker is sending data, a restart will only be possible if the instructions IEC DCL and IEC IFC are executed at the beginning of the program.

IEC instruction

Addressing of Devices as Listener

Syntax: `[n] IEC[z]LAD b`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1
b: address (0 to 31) (decimal value of bits b_1 to b_5 of IEC address)

Remarks: Devices addressed as listeners accept data from the IEC bus. Several listeners can be addressed simultaneously so that the same information can be transferred simultaneously to several devices.

The address b is always specified as the decimal value of address bitsb_1 to b_5 of the data bus. In line with the standard, the same listener and talker addresses may be assigned to a device. In this case bits b_1 to b_5 of the address are identical.

The statement outputs the listener addresses 32 to 63 (decimal) together with ATN.

IEC instruction

Local Lockout

Purpose: This instruction is used to inhibit the manual operation of all devices. Even the key "Go to local" usually present on the device is no longer effective. The instruction is used to protect against manual maloperations during IEC-controlled sequences.

Syntax: `[n] IEC[z]LLO`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Remarks: The instruction can be reset as follows:

a) Only temporarily (up to next addressing as listener) with

`IEC zLAD b` and `IEC zGTL`

b) Completely with

`IEC zNREN` and `IEC zREN`

The statement outputs the control character 17 (decimal) together with ATN.

Note: Resetting requires that the interface functions on the devices are in line with the standard. This is not always the case. Some devices (contrary to the standard) still accept manual control after IEC DCL.

IEC LISTON
IEC LISTOFF

IEC instruction

Parallel Output on the IEC Bus

Purpose: The instruction IEC LISTON outputs the screen contents onto the IEC bus. It is useful for example in outputting to an IEC-bus printer or in the case of program or data transfer to another processor.

Syntax:

```
[n] IEC[z]LISTON b
[n] IEC[z]LISTOFF
```

n: line number, also including label
z: Number of IEC bus 1 or 2, default 1
b: address of output devices (0 to 31)

Remarks: Output to the IEC bus will be continued until the instruction IEC LISTOFF is executed. Subsequently, PRINT, INPUT and LIST are output only on the screen.

The LISTOFF instruction deaddresses the listener with UNL.

Example: Transfer of the BASIC program to a printer with the IEC address 10:

```
IEC LISTON 10
LIST
IEC LISTOFF
```

Note: At the beginning of a listing (command LIST) the Escape sequence "Ec[x" is transmitted, which for example informs the connected printer that the following Escape sequences should not be executed but included in the listing.

IEC instruction

Deaddressing of Devices

Purpose: If a different device or a different group of devices is to be addressed, the previously addressed devices must be deaddressed.

Syntax: `[n] IEC[z]MTA` Deaddressing of the talker

Synonym: `[n] IEC UNT`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Remarks: The statement outputs the control character of the own talker address together with ATN. The own address is set by the IEC ADR instruction, the default value is the talker address 31 corresponding to the decimal character 95 (decimal).

IEC instruction

Output of Strings with Addressing

Syntax: `[n] IEC[z]OUT a1 [:a2] , s$ [;]` (full instruction)

n: line number, also including label
z: number of IEC bus 1 or 2, default 1
a_1: listener address (0 to 31) *)
a_2: secondary address (0 to 31)*)
s$: data to be output
;: CR, LF is transmitted if the semicolon or the comma is missing.

The individual steps of the instruction are:

a) Address device with listener address a. (LADa)
b) A secondary address (a) is transmitted if it has been specified. (SADa)
c) Transmission of own talker address (MTA)
d) Transmission of character string s$, possibly with EOI (see IEC EOI) at the last character
e) Deaddressing of device (UNL).

Note: A delimiter should generally be transmitted with this data output with which the listener recognizes the end of the data output. The manual for the device description indicates the delimiter to which the device responds. The ASCII characters LF or CR and LF are commonly used.

*) Decimal value of bits b_1 to b_5

IEC $OUT
IEC %OUT

IEC instruction

Output of String Variables without Addressing

As controller and talker/listener:

Syntax: `[n] IEC[z]$OUT s$`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1
s$: character string to be output

Output of a single character:

Syntax: `[n] IEC[z]%OUT b`

z: number of IEC bus 1 or 2, default 1
b: numerical expression (0 to 255) for the character to be output.

Remarks: The device can only act as a talker with both instructions. It sends either the string variable s alone or a single character which has the decimal equivalent b. Characters may only be sent to previously addressed listeners.

When the computer has released control it will as talker/listener transmit the information with these instructions.

Example: Output of bytes to a device with the address 6:

```
50  IEC NEOI
100 N$="N0"
110 IEC LAD 6
120 IEC $OUT "A1J1"+N$
130 IEC %OUT 13
140 IEC UNL
```

100: String variable N$ receives the content N0.
110: Device with the address 6 addressed as listener
120: String variable "A1J1N0" sent.
130: Delimiter CR sent
140: Unlisten

Note: If BASIC is in the status of active controller with these instructions, the controller will first transmit its own talker address (MTA).

The last byte of the string variable may be transmitted with EOI (see IEC EOI).

IEC instruction

Parallel Poll Configure

Purpose: This instruction replaces the instructions IEC [a] LAD b, PPC, PPE k_1, k_2, UNL (see there and Section "Parallel Poll").

Syntax: `[n] IEC[z]PCON b, k1, k2`

n: line number, also including label
z: IEC bus number 1 or 2, default 1
b: address (0 to 31)
k_1: sense bit (Sb, 0 or 1)
k_2: 1 to 8 = DI01 to DI08 as reply line

Remarks: IEC-bus compatible devices may be equipped with the capability of registering their status on one of the eight data bus lines DIO1 to DIO8. Up to eight devices can participate in this parallel poll, each device being assigned its own DIO line.

IEC instruction

Parallel Poll Configure

Syntax: `[n] IEC[z]PPC`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Remarks: IEC-bus-compatible devices can be equipped with the capability of signalling their device status on one of the eight data bus lines DIO1 to DIO8. Up to eight devices can participate in this parallel poll, a single DIO line being assigned to each device.

The IEC PPC instruction initiates an instruction sequence which determines the allocation of the device to the data bus lines.

The instruction outputs the control character 5 (decimal) together with ATN.

Note: Execution of a parallel poll is described in Section 1.

IEC instruction

Parallel Poll Disable

Purpose: This instruction terminates the polling of only these devices currently addressed as listeners.

Syntax: `[n] IEC[z]PPD` (parallel poll disable)

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Remarks: The statement outputs the control character 112 (decimal) together with ATN.

Note: The execution of a parallel poll is described in Section 1.

Related instructions: `IEC PPL, IEC PPC, IEC PPE, IEC PPU, IEC PCON`

IEC instruction

Parallel Poll Enable

Purpose: This instruction determines on which DIO line the device is to reply in the case of parallel poll and whether this line is 0 or 1. The figure k_2 is assigned to one of the 8 lines and k_1 may invert the active status 0 or 1 determined by the manufacturer.

Syntax: `[n] IEC[z]PPE k1 k2`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1
k_1: sense bit (0 or 1)
k_2: 1 to 8 = DIO1 to DIO8 as the reply line

Example:

```
100 IEC PPE 14
```

Line 4 is assigned.

IEC instruction

Triggering of Parallel Poll

Purpose: Following execution of the IEC PPL instruction, all devices previously set for polling with IEC PPC and IEC PPE will transmit their status reply.

Syntax: `[n] IEC[z]PPL v%` (status reply)

n: line number, also including label
z: number of IEC bus 1 or 2, default 1
v%: integer variable into which the status reply of all devices is entered.

Remarks: This instruction simultaneously sends the messages EOI and ATN and reads the status word on the bus into the variable v%.

IEC instruction

Parallel Poll Unconfigure

Purpose: The parallel poll is a way of checking by the controller whether the connected devices are ready. A specific instruction sequence is necessary which is described in Section 1. The parallel poll status of all devices is terminated by the universal instruction IECPPU.

Syntax: `[n] IEC[z]PPU`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Remarks: This universal instruction terminates the capability of all devices to reply to a parallel poll. A new setting may be subsequently made if necessary.

The statement outputs the control character 21 (decimal) together with ATN.

IEC instructionen

Line Messages

Purpose:

The controller provides the facility for also controlling the management lines via BASIC instructions (except the line "SRQ" which is not controlled by the controller). This, however, is only necessary with special programming functions, since the PCA5 automatically controls the management bus with all IEC instructions as prescribed in the standard.

The line message REN permits to operate the connected devices via the IEC bus. However, the device will enter the 'remote mode' only following its receiving a listener address.

Syntax:

Line message

```
[n] IEC[z]REN      Line "REN" active.

[n] IEC[z]NREN     Line "REN" passive.
```

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Remarks:

IEC REN may only be transmitted if the controller has the system control.

Possible error:

REN does not go low (true)

Possible reason:

Jumper not plugged in on system controller.

IEC instruction

Release of Control

Purpose: If several controllers are to be operated on one bus, only one of them may have the bus control at a time. The first IEC statement of all other controllers must be IECRLC (release control). The computer will then be a talker/listener on the bus once the instruction has been executed.

Syntax: `[n]IEC[z]RLC`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1
default value or no data: 1

Remarks: All stored IEC parameters are deleted by a software reset with IEC RLC and are subsequently reentered. Transmission of IFC and REN is prohibited following IEC RLC and leads to the corresponding error messages.

Possible error message: *not an IEC-bus controller*

Reason: The computer has already been talker/listener

IEC instruction

Send Service Request

Purpose: If the computer has given up the control of the bus with IEC RLC, it may as talker/ listener send a service request at any time calling for service from the controller. The service request may be used for example to obtain control again.

Syntax: `[n]IEC[z]RQS b`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1
b: status variable to be output to the bus during serial poll 1 to 255

Remarks: During the execution of the IEC RQS instruction, the SRQ line of the management bus is at first set to low (i.e. true). On the appearance of an SRQ, the controller carries out a serial poll. With the controller being addressed as a talker in the serial poll, it issues the status variable b to the bus. Bit7 (dec. 64), i.e. RQS message, is high. The controller cancels the service request only after the status byte has been read. The SRQ line goes high as long as no service request is raised by another device. If no service request is found in the serial poll of the controller, bit 7 (RQS) is set to low.

If a service request is detected in the parallel poll of the controller, the assigned bit for the parallel poll reply is set high.Correspondingly, this bit is set to low when no service request is found.

Example: Send service request

```
IEC RLC, ADR 11, RQS 10
```

Request return of control

```
100 IECRLC,ADR(15)
.
.
.
500 IECRQS(15)
510 IECWTCT
```

Possible error message: *not a IEC-bus talker/listener*

Reason: The computer has the control of the bus and therefore cannot send an SRQ.

Note: The cancellation of the SRQ message is performed in line with the laid down standard only for serial poll and not for parallel poll.

IEC instruction

Secondary Addressing of Devices

Purpose: Transmission of a secondary address (associated listener or talker must already have been addressed).

Syntax: `[n] IEC[z]SAD b`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1
b: address (0 to 31) (decimal value of bits b_1 to b_5 or IEC address)

Remarks: Secondary addresses may be transmitted to both listeners as well as talkers which the devices can then interpret on their own. The device functions defined by the manufacturer can thus be activated in devices equipped with this function.

This statement outputs the control characters 96 to 127 (decimal) together with ATN.

IEC instruction

Selected Device Clear

Purpose: The instruction sets the addressed listeners into the basic status (as determined by the manufacturer).

Syntax: `[n]IEC[z]SDC`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Remarks: This statement outputs the control character 4 (decimal) together with ATN.

IEC instruction

Setting the Device Characteristics

Purpose: The waiting time of the time control, the input terminator, the setting or non-setting of the EOI line and the output terminator are device specific.

The IEC instruction determines the address and thus the peripheral device these characteristics are valid for, which are then stored in the device driver. All following inputs/outputs , carried out using the IEC IN and IEC OUT instructions e.g., use these parameters.

Syntax: `[n] IEC[z]SET a,b,c,d`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1
a: talker/listener address 0 to 31 (see IEC LAD, TAD, IN, OUT)
b: time-out monitor 0,1 to 32767 (see IEC TIME)
c: input terminator 0,1,2 to 255 (see IEC TERM)
d: output terminator without (with) EOI 0 (1) (see IEC EOI)

Remarks: These characteristics are set for all following IEC inputs/outputs using the instructions
IEC TIME
IEC TERM
IEC (N) EOI.

WHEN the IEC SET instruction is used these characteristics only apply for one defined address. For all addresses not defined by IEC SET the parameters set with the IEC TIME, TERM, EOI instructions are still valid.

Assignment of all 31 addresses is possible, (however, max. 15 devices may be connected) thus enabling a clear description of the device characteristics in the program.

Example:

```
IEC SET 8,1000,1,1
IEC SET Uds,1000,1,1        '1 sec timeout
                            ' EOI for EOS on input only
                            'EOI with last character on output
```

IEC instruction

Serial Poll Disable

Purpose: This instruction terminates the serial poll. The devices will then reply with their associated data and no longer with their status byte.

Syntax: `[n] IEC[z]SPD` (serial poll disable)

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Remarks: This statement outputs the control character 25 (decimal) together with ATN.

IEC instruction

Serial Poll Enable

Purpose: This instruction initiates a serial poll. All devices equipped with this function will then reply with their status byte after being addressed as a talker (with IEC TAD a). The data is entered into the controller with IEC %IN v%. The addressing and data inputs are repeated until all devices have been polled.

Syntax: `[n]IEC[z]SPE`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Remarks: This statement outputs the control character 24 (decimal) together with ATN.

IEC instruction

Serial Poll

Purpose: The serial poll of a device is carried out using this instruction. The individual steps of the instruction are:

a) Enable serial poll (SPE)
b) Address with talker address b ($TADb_1$, $SADb_2$)
c) Read in device status in v% (IEC% IN v%)
d) Deaddress talker (MTA)
e) Disable serial poll (SPD)

Syntax: `[n] IEC[z]SPL b1 [; b2], v%`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1
b_1: Talker address (0 to 31) (decimal value of IEC address)
b_2: Secondary address (0 to 31)
v%: fixed decimal point variable into which the device status is read.

Note: If an error, e.g. timeout, occurs within the SPL sequences since the addressed device is not present, the bus will remain in an undefined status. IECSPD must be run in any case. This instruction may be incorporated into an error handling routine.

IEC instruction

Set Time T1 on the Bus

Purpose: For the testing of IEC-bus systems and for obtaining the maximum data rate, the time T1 as laid down in the IEC-625 standard can be set in the computer.

Syntax: `[n] IEC[z]T1 b`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1
b: parameter between 0 and 7 (1.75 µs). Default 500 ns.

Remarks: T1 defines the time between applying data on the bus and setting DAV true, while a byte is transferred from the controller to the IEC bus. T1 is set in steps of 250 ns.

Note: T1 = 0 or 1 is outside the specification of the IEC standard.

b	T1
0	0
1	250 ns
2	500 ns
3	750 ns
4	1 µs
5	1,25 µs
6	1,5 µs
7	1,75 µs

IEC instruction

Addressing of Devices

Purpose: Addressing a talker

Syntax: [n] IEC[z]TAD b

n: line number, also including label
z: number of IEC bus 1 or 2, default 1
b: address (0 to 31) (decimal value of bits b_1 to b_5 or IEC address)

Remarks: A device addressed as a talker transmits its data on the IEC bus when the ATN line has been reset. Only one talker can be addressed at one time. Addressing of a second talker automatically deaddresses the first (in line with the standard).

The address b is always specified as the decimal value of the address bits b_1 to b_5 of the data bus. In line with the standard, a device may be assigned the same talker and listener address. Bits b_1 to b_5 are identical in this case.

This statement outputs the talker addresses 64 to 95 (decimal) together with ATN.

IEC instruction

Take Control

Purpose: Activities on the IEC bus are usually controlled with a single controller. Sometimes two or more controllers participate in the control. The IEC bus standard includes the requirements for this case.

Two interface functions are required for the control:

a) **System control**

This function can transmit the messages "Interface clear" (IFC) and "Remote enable" (REN) at any time and must be performed by the same controller throughout the entire period of bus operation.

b) **Control function**

This function enables the controller to transmit addresses, instructions and data and to carry out device polls. The control function may also only be performed by one controller at a time but may also be transferred to another controller. BASIC has the following instructions for this purpose:

Syntax: `[n] IEC[z]TCT` (take control)

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Remarks: Transfer is carried out using the following instruction sequence:

```
IEC TAD     Addressing of controller which is to take control
IEC TCT     Transfer of control
```

This statement outputs the control character 9 (decimal) together with ATN.

IEC instruction

Definition of the Input Terminator

Syntax: `[n] IEC[z]TERM b`

n:	line number, also including label
z:	number of IEC bus 1 or 2, default 1
b = 0:	CR and LF or EOI
b = 1:	only EOI
b = 2...255:	decimal value of the terminator in ASCII code (table 6-7) (or EOI)

Remarks: For data inputs, a terminator must be transmitted which the computer recognizes as the end of the input. The instruction is as follows if the input is to be terminated with e.g. "LF": IEC TERM 10. The input is terminated by IEC TERM 0 after the character combination "CR" and "LF" has been transmitted.

Independent of the terminator specified by the instruction IEC TERM, data input will always be terminated if the line message "END" (on EOI) is transmitted. The instruction will be IEC TERM 1 if only the message "END" (and not one of the ASCII characters transmitted on the lines DIO1 to 8) is to terminate the input.

If the terminator does not change, it will be sufficient to include the instruction only once in the program before the first data input.

Note: IN instructions such as IEC INa and IEC $IN do not load the terminator into the string variable.

IEC instruction

Time-out Monitor

Syntax: `[n] IEC[z]TIME b`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1
b: time in ms (1 to 32767)
0 = switch off time-out monitor

Remarks: The controller provides the facility for monitoring the time during a handshake procedure. The time required to transfer a character is monitored.

In line with the IEC standard this will be the time from 5 to 7 if the controller operates as a talker (output handshake) or from 4 to 5 if the controller is a listener (input handshake)

The controller waits for a period of b [ms] with a handshake. The instruction should be written at the start of a program if devices are connected which require more than 1 s for character transfer. Without this instruction, a value of 1 s will apply.

Possible error message: The controller will terminate the handshake if the specified waiting time is exceeded. The following error message will appear.

ERROR 10: "IEC bus time out"

The program is terminated following output of the error message.

IEC instruction

Unaddressing of Devices as Listener

Purpose: The device or the group of devices previously addressed as listener is reset into the passive status (unaddressed) using this instruction.

Syntax: `[n] IEC[z]UNL`

n: line number, also including label
z: number of IEC bus 1 or 2, default 1

Remarks: The controller sends the character 63 (3FH), i.e. the listener address 31 (1FH) together with ATN.

In the case of the higher IEC instructions such as IEC IN and IEC OUT, this message is automatically sent by the controller in order to set the devices connected to the IEC bus into a defined status.

IEC WMLA
IEC WMTA

IEC instruction

Wait for Addressing

Purpose: The controller must be addressed prior to data transfer before it can operate as a talker/listener on the IEC bus. In order to synchronize data transfer, the instrument can wait until its address specified in the IEC ADR instruction is received. This is effected using the instructions

IEC WMLA (wait my listener address) or
IEC WMTA (wait my talker address)

Syntax: `[n]IEC[a]WMLA [n]IEC[z]WMTA`

n: line number, also including label
z: number of the IEC bus 1 or 2, default 1

Remarks: The running program waits on receiving the instruction IEC WMLA or IEC WMTA until the address of the instrument appears together with ATN. All other data and commands on the bus are ignored. The instruction does not have a time-out function to synchronize infrequent data transfers.

Example: Operation as a listener on the bus

```
100 IEC RLC
110 IEC TERM13:IEC ADR10
120 IEC WMLA
130 IEC $INA$
```

Operation as talker on the bus

```
100 IEC RLC
110 IEC ADR12
130 IEC WMTA
140 A$="DATEN"+CHR$(13)
150 IEC $OUT A$
```

Note: Operation of the PCA as a talker/listener on the bus is described in Section 1.5.

IEC instruction

Wait for Transfer of Control

Purpose: If the computer is operated as a talker/listener on the bus but is to control the bus transfer, it can wait for transfer of the control by means of the IECWTCT instruction (wait take control).

Syntax: `[n] IEC[z]WTCT`

n: line number, also with label
z: number of IEC bus 1 or 2, default 1

Remarks: The program waits on receiving this instruction until the computer is addressed with its talker address specified in the IECADR instruction and receives the message TCT. The computer controls the bus once the instruction has been executed and can address connected devices and handle data transfer. The IECWTCT instruction does not have a time-out function.

Example: Wait for transfer of control

```
100 IEC RLC
.
.
.
1000 IEC WTCT
1010 IEC OUT 16,"DATA"
```

Instruction (structure element)

Changing the Program Run

Purpose: Changes the program run according to the result of a numeric expression.

Syntax 1:

```
[n]  IF a GOTO n
[n]  IF a THEN n
[n]  IF a GOTO Label
[n]  IF a GOSUB n
[n]  IF a GOSUB Label
```

n: line number, also including labels
a: expression

Remarks: If the expression is true (i.e. not 0), e.g. because the comparison is fulfilled, the program will be branched. If the expression is false (i.e. 0), the program will be continued with the instruction following IF.

Syntax 2:

```
[n] IF a THEN instruction [:instruction]...
[n] IF a THEN instruction ELSE instruction
                          [:instruction]...
```

Remarks: THEN may be followed by a further instruction or a series of instructions separated by ":". In particular, nesting of further IF instructions is possible.

If the expression is true, the instruction following THEN will be executed until the end of the line or the keyword ELSE. If the expression is false, the keyword ELSE will be searched for and the instruction(s) following it executed. If no ELSE is found, the program will be immediately continued with the next line.

Example:

```
100 IF A$="JA" THEN PRINT "OK":GOSUB 300
200 IF F AND A=3 THEN RETURN
300 IF A THEN GOSUB Abc ELSE GOSUB Bcd
```

Instruction (structure element)

Syntax 3:

```
[n] IF a THEN
[n]       instruction
          .
          .
          .
[n]       instruction
[n] ENDIF

oder

[n] IF a THEN
[n]       instruction
          .
          .
          .
[n]       instruction
[n] ELSE
[n]       instruction
          .
          .
          .
[n]       instruction
[n] ENDIF
```

Remarks: If nothing follows the keyword THEN in the line (except an instruction separated by ':'), this IF instruction will be interpreted as structure element.The following lines will be executed as a block if the expression is true until the instruction ELSE or ENDIF is found. The ELSE instruction may be used but, in any case, the ENDIF instruction must mark the end of block for otherwise an error message will be output. If the expression is false, the ELSE instruction will be searched for and, if available, the lines between ELSE and ENDIF executed. Further structures may be nested in an IF-THEN-ENDIF structure. In these cases, better readability should be ensured by indenting lines.

Example:

```
100 IF A<1 THEN
110    Print "OK"
120    GOSUB Subr1
130 ELSE
140    Print "ERROR"
150    GOSUB Subr2
160 ENDIF
```

Possible error message: *ERROR 48:"no IF-struc.,[ELSE], ENDIF match"*

Reason: THEN/line indentation has been found without ENDIF, or ELSE or ENDIF has been found without active IF-THEN/end of line instruction.

Related instructions: ELSE, ENDIF

Instruction

Keyboard Read-in

Purpose: The INKEY instruction allows data to be read into the controller from the keyboard. In contrast to the INPUT statement, it does not stop the program and can only read in one character at a time.

Syntax: `[n] INKEY v$`

n: line number, also including label
v$: string variable

Remarks: With each INKEY statement, the controller stores the last character from the keyboard buffer in the variable and removes the character from the keyboard buffer.

Related instructions: `INPUT#, INPUT$()`

Example:

```
100 INKEY C$
```

Wait until any key is pressed (use of an empty character string)

```
200 INKEY A$: IF A$="" THEN 200
```

Numeric function

Reading and Writing via I/O Addresses

Purpose: To read and write via the I/O addresses

Syntax: `INP(a)` Function

a: input address (0 to 65535)

Remarks: The I/O addresses 0 to 65535 can thus be accessed to. 8-bit words can be read in or out for these addresses using INP or OUT.

Related function: `OUT`

Example: Read in I/O address 4

```
100 PRINT INP(4)
```

Note: With the PCA2/5, this function addresses interfaces connected to the I/O bus. With the PCA12/15, however, the I/O addresses of the multibus are addressed via the 286 CPU. This is another reason why the BASIC instructions or MS-DOS system calls should be used for all inputs and outputs instead of accessing the interfaces directly.

Instruction

Keyboard Input

Purpose: The INPUT statement is suitable for entering data or string constants from the keyboard into the controller during a program.

Syntax: `[n] INPUT["t";]v_i[,vn]...`

n: line number, also including label
t: string constant output on the screen by the input
v: numeric or string variable

Remarks: The string constant (if entered) is output on the screen in the case of the INPUT statement. The controller then waits with a flashing cursor for data input. This is terminated by the return key. In addition, a question mark is displayed without "t".

The value of the variables will be set to 0 if return is entered without a character.

If several variables v_i are present after INPUT, these can be read in all together if they are each separated by a comma, or individually with return.

Related instructions: `INPUT#, INPUT$()`

Example:

```
100 INPUT F
200 INPUT A,B,C$
```

Display: ?□

Program branching:

```
1000 INPUT "AGAIN";A$
1010 IF A$="YES" THEN 100
```

Display: *AGAIN* □

Possible error message: *ERROR 37: "variable type mismatch"*

Reason: An attempt has been made to load a text into numeric variables.

Instructions

Note:

When the INPUT instruction is reached, entries made via the keyboard are output on the monitor. Entries to the INPUT line are not made via the keyboard, but by using the return key the contents of the screen memory are read to the line where the cursor is positioned.

It is thus possible to make entries of the complete screen by the vertical shift of the cursor and thus for example a menu control can be set up with the PRINT instructions.

If the INPUT instruction is written without a text, the content of a line is read from the second place with reference to the left-hand screen edge. With a text having n places in the INPUT line, the content of a line is read from the n + 1th place.

Example:

```
100 PRINT "1234567"
110 INPUT A$
```

If the flashing cursor is positioned on the PRINT line and the return key is pressed to terminate the INPUT line, the value of A$ will be:

```
234567
```

```
100 PRINT "1234567"
110 INPUT "TEXT";A$
```

Value of A$: 567

Instruction

Read-in from a File/Interface

Purpose: Data can be read from a file selected using the OPENI# instruction or an interface and assigned to a numeric or string variable.

Syntax: `[n] INPUT#a, v1[,v2]...`

n: line number, also including label
a: channel number (1 to 15)
v_i: string variable to be read into

Remarks: Up to 80 characters can be read into the numeric variable or the string variable v_1 to vi.

Leading spaces and carriage return are ignored in the case of numeric variables. A space, carriage return or a comma is interpreted as the end of a number. Only the carriage return character is recognized as a delimiter in the case of character strings.

Related instructions: `INPUT$(), INPUT, OPENI#, CLOSE#`

Example:

```
10 INPUT#5,C$
```

Read data from floppy disk

```
100 OPENI#1,"DATEI.ASC"
110 INPUT#1,A
```

Possible error message: *ERROR 39: "out of data"*

Reason: An attempt has been made to read into more data than are present in the file.

Note: The following error message will appear if an instruction is used when the drive has no floppy.

"DOS: drive not ready"

String function

Read-in from File or Interface

Purpose: Read-in of a string of n characters which has been read from the keyboard or from an interface/file with the channel number.

Syntax: `INPUT$ (vn,[#a])`

vn: number of characters to be read (1 to 65535)
a: Channel number; read-in from keyboard if not further specified.

Remarks: If the keyboard is used for the input, no characters will be displayed on the screen. The program is continued only when all the n characters have been entered.

n characters are read in from a file or interface unless the file is shorter. End of file is recognized by the fact that the number of returned characters LEN(v$) is not equal to the number of characters requested.

Single characters are read in with special case n = 1.

All characters are accepted. In particular, in contrast to the INPUT# instruction, all control characters, such as carriage return, are accepted.

Related instructions:: `OPENI#, CLOSE#. INPUT#, INPUT`

Example:

```
10 OPENI# 3,"DATA"
20 REPEAT
30   A$=INPUT$(1,#3)
50   PRINT RIGHT$(HEX$(ASC(A$)),2);" ";
70 UNTIL LEN(A$)=0
80 CLOSE# 3
```

Reads single characters from a file and converts them to a 2-digit HEX number for output on the screen.

Numeric function

Integer Function

Purpose: Reads in the greatest integer which is equal to or less than a.

Syntax: `INT(a)`

a: constant, variable or numeric expression

Remarks: If a is positive, the digits after the decimal point are set to 0, if a will be negative, the number will be rounded to the next integer.

Example:

```
100 C=INT(A*B)
200 A=INT(A+.5)

PRINT INT(1.6) →1
PRINT INT(-1.4) →-2
```

Graphics instruction

Invert Graphics

Purpose: The INVERT instruction inverts the graphic display for the whole of the screen. All blanked dots become unblanked and the unblanked dots become blank. The instruction may be repeated as often as required. The output on the screen is inverted each time.

Syntax: `[n] INVERT`

n: line number, also including label

Remarks: The INVERT instruction does not change the contents of the graphics memory but only switches over the screen output. This type of output is for example not conveyed to the printer.

Note: This instruction is only relevant for b/w monitors. With the color graphics option PCA-B3, the display on the screen is determined by the color instruction.

Related instructions: `SET, COLOR`

Graphics Instruction

Labelling of Graphics

Purpose: Using this instruction, the specified character string is used for labelling purposes in graphics mode.

The current position of the imaginary pen determines the top left edge of the labelling.

Syntax:

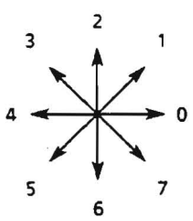
```
[n] LABEL s$ [,a[,b[,c]]]
```

n: line number, also including label
s$: string for labelling
a: 0 to 15 size of character
b: 0 to 7 writing direction
c = 0: bright on dark background
c = 1: dark on bright background

Remarks: Non-specified parameters a, b, c are interpreted as 0.

Note: The LABEL instruction operates analogous to the graphics instructions. So as not to destroy existing graphics all dots can be inverted with a previous SET-1.

Parameter b:
Writing direction



Example:

```
100 LABEL "<--3.25m-->"
200 LABEL "PCA",15,1,1
```

String function

Separate First Character from a String

Purpose: A certain number of characters can be removed from the left side of a character string for further processing using the LEFT$ function.

Syntax: `LEFT$(s$,a)`

s$: character string variable or constant
a: number of characters

Remarks: The parameter a specifies the number of characters to be used for further processing. The complete character string will be used if the number of characters amounts to less than a. The output character string s$ remains unaffected by this operation.

Related functions: `RIGHT$, MID$`

Example:

```
100 A$="PROCESSCONTROLLER":PRINT LEFT$(A$,7)
```

Display: *PROCESS*

Numeric function

Length of a String

Purpose: The LEN function supplies the number of characters in a string variable (e.g. A$) or the length of a string constant so as to continue calculation with this number.

Syntax: `LEN (s$)`

s$: String variable or string constant in quotation marks

Remarks: All character, even spaces, are counted when calculating the lenght. The result can be used as a numeric variable and assigned or output accordingly.

Example:

```
?LEN ("CONTROLLER")
```

Output on screen: *10*

```
100 G=LEN (A$)
```

Instruction

Assignment of Variables

Purpose: For reasons of compatibility, the LET instruction forms part of the standardized ANSI BASIC in the PCA BASIC. It may be left out when assigning variables.

Example: Instead of

```
100 LET A=10
```

```
100 A=10
```

may be used.

Command

Display of an Entered Program on the Screen

Purpose: The LIST command is suitable to output a BASIC program entered via the keyboard or loaded from the floppy disk on the screen.

Syntax: `LIST [n] [-[m]]`

LIST n-m output from line n to line m
LIST n- output from line n up to end of program
LIST -m output from start of program up to line m
LIST n output line n

Remarks: With this command it is possible to enter line numbers from which, or up to which, the program is to be output.

Listing is terminated by pressing the Break key.

Note: Listing is possible on the printer using PLIST. Listing may be stopped using the Ctrl (Strg) keys together with S.

Related command: `PLIST`

Instruction

Loading Programs

Purpose: This instruction loads the program from the floppy disk or the fixed disk into the main memory. It is important to exactly specify the program name, including all characters, spaces and extensions for otherwise the program cannot be found. If the program name is written without an extension, BASIC will search for a file with the extension ".BAS.". If strictly a file without extension is to be loaded, the program name must end with ".". The LOAD instructions may be executed directly or under program control.

Syntax: `[n] LOAD s$ [,R]`

n: line number, also including label
R: automatically starts the loaded program (RUN)
s$: program name, also with drive and search path

Remarks: Contrary to ALOAD, the existing program is deleted with LOAD. All variables are also deleted. Programs can therefore be joined together in particular under program control.

Related instructions: SAVE, ALOAD

Possible error message: *ERROR 52: "DOS open error"*

Possible Reason: Program name does not exist.

Instruction

Loading (Machine) Routines

Purpose: LOAD# can be used to load (machine) routines which are to be called using CALL#.

Syntax: `[n] LOAD# a, s$`

n: line number, also including label
a: numeric expression, value 1 to 7, link number
s$: filename, also with drive and search path

Related instruction: `CALL#`

Numeric function

Logarithmic Function

Purpose: This function calculates the natural logarithm (to base e = 2.7182818).

Math.: $y = ln\, x$

Syntax: `LOG(a)`

a: constant, variable or numeric expression

Related function: `EXP`

Example:

```
100 Y=LOG(X)
```

Logarithm to base 10 can be evaluated:

Math.:

$$y = lg\, x = \frac{ln\, x}{ln\, 10}$$

```
100 Y=LOG(X)/LOG(10)
```

Possible error message: *ERROR 31 in LINE 100: "numeric overflow"*

Reason: Argument a ≤ 0

String function

Read Character from the Middle of a String

Purpose: Reads from a string a number of b characters from position a. It can be used e.g. to suppress the header in IEC IN instructions.

Syntax: `MID$(s$,a,b)`

s$: string variable or string constant
a: position where reading is started
b: number of characters

Related function: `LEFT$, RIGHT$`

Example:

```
100 A$="PROCESSCONTROLLER":PRINT MID$(A$,8,7)
```

Display: *CONTROL*

Graphics instruction

Moving the Graphic Cursor

Purpose: This instruction moves the graphic cursor to the point of intersection of the x and y coordinates without producing a visible trace. It determines the point where drawing is started.

Syntax: `[n] MOVE x,y`

n: line number, also including label
x,y: numeric expressions for the X/Y coordinates

Related functions: `RMOVE, DRAW, RDRAW, DOT`

This page has been kept free on purpose. The BASIC instruction of an option may be inserted here. The pages to be inserted are to be found in the manual of the option concerned.

Command

Command to Delete the Program

Purpose: The complete program and all variables are deleted by the NEW instruction. A new program can then be entered.

NEW resets BASIC to the state after loading.

Related instructions: LOAD, CLR, ERASE, DELETE

Instruction

Loop Instruction (End)

See FOR instruction

Instruction

Branch on Error

Purpose: With this conditional jump instruction a jump is made to an error routine to prevent a program abort in the event of a fault.

Syntax:

```
[n] ON ERROR GOTO m or label
[n] ON ERROR GOSUB m or label
[n] ON ERROR RETURN
```

n: line number, also including label
m: line number to be jumped to or line number of subroutine
label: any character sequence consisting of letters, digits and underline characters

Remarks: If GOTO/GOSUB is not followed by a line number but by a character sequence starting with a letter, BASIC will search for the line of this label. It is to be noted that this label is located directly after a line number and must end with a colon.

This instruction may be present at any position in the program but must be read in at least once before an error occurs. If this instruction is used several times, the program will jump to the position specified by the last ON ERROR GOTO m instruction.

After branching on occurrence of an error, the ON ERROR instruction is inactive until set once again with a new ON ERROR GOTO instruction. The setting can also take place in the subroutine and should be the last executable instruction prior to RETURN (nesting depth).

This branch into an error routine can be set inactive again using ON ERROR RETURN. Following this instruction, error messages are again displayed on the screen.

In the error-handling routine, the error line can be polled with the function ERL(0) and the error number with ERM(0).

Related functions: ERL, ERM

Instruction

Branch into Subroutines

Purpose: A branch into subroutines can be made with ON...GOSUB depending on an expression.

Syntax:

```
[n] ON a GOSUB m1[,mi]...
[n] ON a GOSUB label, [,labeli]...
```

n: line number
m_i: branch line numbers
a: numeric expression
label: any character sequence consisting of letters, digits and underline characters

Remarks: RETURN is used at the end of the subroutines just as with the GOSUB statement. See also ON GOTO.

If GOSUB is not followed by a line number but by a character sequence starting with a letter, BASIC will search for the line of this label. It is to be noted that this mark is located directly following a line number and must end with a colon.

Related instructions: `RETURN, ON GOTO`

Example:

```
100 ON R GOSUB1000,2000
110 END
1000 REM Subroutine R=1
1100 RETURN
2000 REM Subroutine R=2
2100 RETURN
```

Possible error message: *ERROR 25: "undefined line or label"*

Reason: Branch line number does not exist

Instruction

Multiple Branching

Purpose: Branches to one of several branch targets, depending on the value of an expression.

Syntax:

```
[n] ON a GOTO m1 [, mi]...
[n] ON a GOTO label1, [labeli]...
```

n: line number, also including label
a: variable, constant or numeric expression
m...mi: line numbers of branch targets
label: any character sequence also with letters, digits and underline characters

Remarks: The program jumps to the specified program lines depending on the magnitude of a, namely:

a < 1 No jump
a = 1 Jump to m_1 or $label_1$
a = 2 Jump to m_2 or $label_2$
:
a = i Jump to m_i or $mark_i$
a > i No jump

A numeric expression may be used for a. Any digits after the decimal point of a are ignored. If GOTO is not followed by a line number but by a character sequence starting with a letter, BASIC will search for the line of this label. This label must be located directly following a line number and end with a colon.

Related instruction: ON GOSUB

Example:

```
100 ON X GOTO200,300,400
200 REM BRANCH TARGET WITH X=1
300 REM BRANCH TARGET WITH X=2
400 REM BRANCH TARGET WITH X=3
```

For the calculated jump:

```
100 ON (A*B)-(C*D) GOTO 200,300,400
```

For extension if line length is insufficient for all branch line numbers:

```
100 ON R GOTO 200,210,220,230,240,250,260,270,280,290
110 ON R-10 GOTO 300,310,320,330,340,350,360,370,380,390
120 ON R-20 GOTO 400,410,420,430,440,450,460,470,480,490
```

Possible error message: *ERROR 25: "undefined line"*

Reason: Branch target does not exist.

Instructions

Event-controlled Branch on Pressing a Key

Purpose: Branches to a subroutine by pressing any key or softkey.

Syntax:

```
[n] ON KEY GOSUB m or label
[n] ON KEY GOTO m or label
[n] ON KEY RETURN
```

n: line number, also including label
m: line number

Remarks: If GOSUB/GOTO is not followed by a line number but by a character sequence starting with a letter, BASIC will search for the line of this label. This label must be located directly following a line number and end with a colon.

An interrupt is generated in the computer by pressing a key. Since, however, current statements must not be interrupted, the interrupt is only detected with the next line number.

The key code can be read in and processed further in the subroutine using INKEY.

The branch target can be redefined in the program run using ONKEY GOSUB. The target last specified is always the valid address. A return from the subroutine is made using the RETURN instruction.

ON KEY RETURN switches off this mode. Pressing of a key no longer causes branching.

Example: Abort program with key E

```
100 ON KEY GOSUB 1000
.
.
.
1000 INKEY A$: IF A$="E" THEN END
1010 ON KEY GOSUB 1000: RETURN
```

Note: Renewed calling is blocked following a branch into the subroutine until a new target is defined using ON KEY. In order to avoid nesting, this instruction should be directly followed by RETURN (i.e. only separated by ':').

Possible error message: *ERROR 25: "undefined line or label"*

Reason: Branch line number does not exist.

IEC instruction

Event-controlled Branching on Service Request

Syntax:

```
[n] ON SRQ [a] GOSUB m or label
[n] ON SRQ [a] GOTO m or label
```

n: line number, also including label
m: line number
a: number of the IEC bus

Remarks: If GOSUB/GOTO is not followed by a line number but by a character sequence starting with a letter, BASIC will search for the line of this label. This label must be located directly following a line number and end with a colon.

The program branches to a subroutine with ON SRQ GOSUB n when a service request of the device is received via the IEC bus interface. The subroutine commences with the line number "n"; the calling device is usually identified by a serial poll and is then serviced.

This instruction may be present at any position in the program but must have been read at least once before being executed for the first time. The last instruction read is executed if the instruction appears several times.

The return from the subroutine to the interrupted main program is made as usual using the RETURN statement. Branching to a subroutine following a service request may be inhibited by the following instruction:

Syntax:

```
[n] ON SRQ RETURN
```

n: line number

Note: A second SRQ remains inhibited after branching (ON SRQ RETURN it will be executed automatically). An ON SRQ GOSUB n line must then be in the program again before BASIC responds to SRQ again. To prevent inadmissible branching as a result of repeated interrupts, the RETURN instruction must be placed immediately after the ON SRQ GOSUB instruction (the two being separated by ':').

```
1000
1010 IECSPL13,V%                          ┐
.                                      Service
.                                      routine
.                                         ┘
1100 ON SRQ GOSUB 1000: RETURN
```

The SRQ service routine must contain at least one SPE command (or one SPL command which executes the SPE command implicitly).

Instruction

Event-controlled Branching

Purpose: If a particular program is to be executed depending on the time, the time and the branch line number can be output in any line using this instruction.

Syntax:

```
[n] ON TIME a GOSUB m or label
[n] ON TIME a GOTO  m or label
[n] ON TIME RETURN
```

n: line number, also including label
m: line number
a: numeric expression indicating the time in hundredths seconds since midnight (max. $100 \times 60 \times 60 \times 24 = 8{,}640{,}000, -1$).

If GOSUB/GOTO is not followed by a line number but by a character sequence starting with a letter BASIC will search for the line of this label. This label must be located directly following a line number and must end with a colon.

The branch line number can be redefined during the program run using ON TIME GOSUB. The line number last specified is always the valid address. A return from the subroutine is made using RETURN.

ON TIMING RETURN inhibts branching to the subroutine.

Note: Renewed calling is inhibited following branching to the subroutine until a new branch target is defined using ON TIME.

Example: Cyclical interrupt (every 100 hundredth seconds, i.e. once a second)

```
10 ON TIME (TIME+100) GOSUB 50
20 GOTO 20: REM ENDLESS LOOP
50 PRINT "SECONDS INTERRUPT"
60 ON TIME (TIME+100) GOSUB 50: RETURN
```

Example: Output of time

```
10 ON TIME((12*60+15)*60+0)*100 GOSUB 200
20 GOTO 20: REM ENDLESS LOOP
200 PRINT "12:15:0 O'CLOCK!"
```

Note: This instruction is only carried out by the PCA, if the real time clock (PCA-B10 option) is fitted.

Instruction

Open a File for Input or Input via Interface

Purpose: Enable the file to read in data (INPUT). Inputs from interfaces, floppy disk and fixed disk take place via the file management of the operating system. Opening a file with this instruction determines that the file is to be read.

Syntax:

```
n OPENI# a,s$
```

n: line number, also including label
a: channel number
s$: file name, also with drive and search path or interface name

Related instructions: CLOSE#, OPENO#, INPUT#, INPUT$()

Example: Open channel 1 in order to read data with the name "DAT1" from the floppy disk:

```
100 OPENI #1,"DAT1"
```

Possible error message: *ERROR 54: "no valid file number", "device driver not loaded"*

Reason: An attempt has been made to open more files than possible, or parameter a lies beyond the permissible range.

Even if the number of files is set >15 in the file CONFIG.SYS, MS-DOS permits only 15 files to be open at the same time. Since BASIC internally requires between 3 and approx. 7 files, the user has only about 8 to 12 open files left.

Possible error message: *ERROR 49: "file already open"*

Reason: An attempt has been made to open an already opened file. The file is closed again with an error message.

Note: The file must be reopened accordingly when changing between read and write functions. A file is opened for output with OPENO#.

Instruction

Open a File for Output or Output via Interface

Purpose: Similar to the OPENI# instruction, the OPENO# instruction is used to open a file for output or an output interface via the file management of the operating system. Opening of a file with this instruction determines that the file is to be written to.

Syntax: `[n] OPENO# a,s$`

n: line number, also including label
a: channel number (1 to 15)
s$: filename, also with drive and search path or interface name

Related instructions: `CLOSE#, OPENI#, PRINT#`

Example: Open channel 1 in order to write data with the name "DAT1":

```
100 OPENO# 1,"DAT1"
```

Possible error message: *ERROR 49: "no valid file number"*

Reason: An attempt has been made to open more files than possible (see also OPENI# instruction).

Possible error message: *ERROR 49: "file already open"*

Reason: An attempt has been made to open an already opened file. The file is closed again with this error message.

Note: The file must be reopened accordingly when changing between read and write functions.

Instruction

Console

CON is the name of the file for console operations.

The console can be serviced by BASIC also with INPUT# and PRINT# instructions.

Syntax:

```
[n] OPENI# a,"CON:"
[n] OPENO# a,"CON:"
```

n: line number, also including label
a: channel number (1 to 15)

Instruction

Standard Input/Output via IEC Bus

Purpose: The standard input/output with the INPUT# and PRINT# instructions or the INPUT$() function is also possible via the IEC-bus interface, in addition and parallel to the IEC instructions. With this programming, all inputs/outputs can be diverted to a different interface, e.g. screen and keyboard for testing purposes. This is done by modifying the OPEN instructions. The disadvantage lies in the fact that the notation is in most cases longer than that of the specific IEC instructions.

Syntax:

```
[n] OPENO#a,"IEC:parameter list"
[n] OPENI#a,"IEC:parameter list"
```

n: line number, also including label
a: file number (1 to 15)

The parameter list may contain the following words and parameters (the notation corresponds to that of the IEC instructions; the functions are described in detail there):

Command	Parameter	Remark
UNI	1 or 2	Determining the interface (IEC1 or IEC2) to which the following commands or inputs/outputs are to be addressed.

Universal commands

Command	Parameter	Remark
LLO	---	LOCAL LOCKOUT, lock manual operation
DCL	---	DEVICE CLEAR, reset device

Addressing commands

Command	Parameter	Remark
LAD	0 to 31	LISTENER ADDRESS, address device as listener
TAD	0 to 31	TALKER ADDRESS, address device as talker
SAD	0 to 31	SECONDARY ADDRESS, secondary address

Addressed commands

Command	Parameter	Remark
GTL	—	GO TO LOCAL, switch selected device to manual operation
SDC	—	SELECTED DEVICE CLEAR, reset selected device
GET	—	GROUP EXECUTE TRIGGER, trigger selected devices

Deaddressing commands

Command	Parameter	Remark
MTA	—	MY TALK ADDRESS (UNT), deaddress the talker
UNL	—	UNLISTEN,deaddress the listener
END	—	sends this character with EOI line active e.g. END 10 sends LF with EOI

Line commands

Command	Parameter	Remark
REN	—	REMOTE ENABLE, enable line
NREN	—	REN, disable REN line
IFC	—	INTERFACE CLEAR, enable line for 500 µs

Multi-controller operating commands

Command	Parameter	Remark
TCT	—	TAKE CONTROL, transfer control to other controller
RLC	—	initialize controller as talker/listener
WMLA	—	wait for addressing as listener
WMTA	—	wait for addressing as talker
WTCT	—	wait for transfer of control

Setting parameters

Command	Parameter	Remark
TERM	0 to 255	Input terminator 0: CR and LF or EOI 1: only EOI (default) 2 to 255: this character or EOI
ADR	0 to 31	set own address, default 31
T1	0 to 31	delay time for line DAV incrementing by 125 ns. Default 3 corresponding to 500 ns.
TIME	0 to 65535	time monitoring in ms 0 corresponds to time monitoring off default 1000 corresponds to 1 s.

Example:

```
300 OPENO# 3, "IEC:LAD1"
310 PRINT# 3, "W3,X3" :REM for UDS5
320 CLOSE# 3
330 OPENI# 4, "IEC:TAD1,TERM10"
340 A$=INPUT$(20,#4)
350 CLOSE# 4
360 PRINT A$
```

is equivalent to

```
310 IEC OUT 1,"W3,X3" :REM for UDS5
330 IEC TERM 10
340 IEC IN 1,A$
360 PRINT A$
```

Instruction

Controlling the Centronics Interface

Purpose: The Centronics interface is serviced via the file management of the operating system.

"LPT" is the name of the file via which the Centronics interface is opened. "PRN" corresponds to the first Centronics interface LPT1.

Syntax:

```
[n] OPENO# a,"LPTz:"
[n] OPENO# a,"PRN:"
```

n: line number, also including label
a: channel number (1 to 15)
z: number of interface 1 or 2

Example:

```
100 OPENO# 1,"LPT1:"
110 PRINT# 1,"PRINTER TEST"
```

After the version 1.90 the device driver LPTx can also read the status with LPTb:

```
100 OPENI#1, "LPT1:"
110 Status = INPUT$(1,#1)
```

or for the second interface:

```
100 OPENI#1, "LPT2:"
110 Status = INPUT$(1,#1)
```

Status = 0: everything o.k.
2: printer: not ready

Related command: FORM

Note: If a lower and upper myrgin is set using the FORM command before the execution of this instruction, a formfeed will be released with the OPEN instruction.

Instruction

Writing via I/O Addresses

Purpose: To write via the I/O addresses of the computer.

Syntax:

```
[n] OUT a,b   instruction
```

n: line number, also including label
a: output address (0 to 65535)
b: value to be written (0 to 255)

Remarks: The I/O addresses between the values 0 and 65535 can be accessed. Using OUT, 8-bit words can be output at each of these addresses.

Note: With the PCA12/15, the I/O addresses of the multibus are addressed via the 286 CPU (see PCA manual). This is another reason why the BASIC instructions or MS-DOS system calls should be used for all inputs/outputs instead of accessing the interfaces directly. In contrast thereto, this instruction is used with the other Rohde&Schwarz controllers to address interfaces connected to the I/O bus.

Related instructions: `INP`

Instruction

Call of Pascal Routines

Purpose: Call of procedures written in Pascal (see PCA-K11 manual).

Syntax:

```
[n] PASCAL a[,v]...
```

n: line number, also including label
a: procedure number of Pascal program
v: integer, floating decimal point or string variable

Example:

```
100 Sweep=3
110 PASCAL Sweep, Startfreq, Stopfreq
```

Numeric function

Reading of Memory Locations

Purpose: Transfers a byte from a memory location.

Syntax: `PEEK(a)`

a: address of the memory location (-32768 to 65535)

Remarks: The parameter a merely sets the address offset. If the segment of the address is outside the BASIC segment when reading memory locations, the SEGMENT instruction must be given first. Integers are stored internally in 2 bytes (MSB = sign). Thus, negative addresses do not represent a negative offset but an offset greater than 32767.

Related instructions: `POKE, SEGMENT, VARPTR`

Example:

```
100 PRINT PEEK(22)
```

Pseudo variable

Circle constant

Purpose: Represents the numerical value of the circle constant π.

Syntax: `PI`

Example:

```
10 A=COS(Grad * PI/180)
20 PRINT PI
   → 3.14159265395
```

Instruction

Generation of Signal Tones

Purpose: Signal tones and tone sequences with variable pitch and duration can be generated using the PCA.

Syntax: `[n] PLAY s$[,a]`

n: line number, also including label
s$: string variable which characterizes the pitch and duration
a: repetition rate (0 to 8; default value 4)

Remarks: Several tones can be included within the string variable. Each tone comprises a letter which characterizes the pitch and a subsequent number for the duration. The symbols # and & prefixed to the tone letter increase or decrease the pitch by a semitone.

The following tones are available:

C D E F G A H B c d e f g a h b

The symbols # and & are used for prefix.

A pause (no tone) can also be generated using the letter P.

The following values can be used for the duration of a tone or pause d:

1 2 3 4 5 6 7 8

where the duration is $\frac{0{,}5\ s}{d}$ in each case

Example:

```
10 A$="G4E4E2F4D4D2C4D4E4F4G4#G1G2"
20 PLAY A$,5
```

Command

Program Output via the Centronics Interface

Purpose: A program can be output on the printer interface using the PLIST command.

Syntax: `PLIST [n] [-[m]]`

PLIST n-m	output from line n up to line m
PLIST n-	output from line n up to end of program
PLIST -m	output from start of program up to line m
PLIST n	output of line n

Remarks: With this command it is possible to specify line numbers which define the start and end of the program output.

Note: For obtaining a page formatted output, see command FORM [m - n].

Related commands: `LIST, FORM`

Instruction

Writing into Memory Locations

Purpose: The following BASIC instruction is used to write a byte to a memory location:

Syntax:

```
[n] POKE a,b
```

n: line number, also including label
a: address of the memory location (-32768 to 65535)
b: value to be written between 0 and 255

Note: Parameter a only specifies the address offset. The address segment can be previously selected using the SEGMENT statement. A negative number as address is not taken as negative offset but as a number greater than 32767.

Random overwriting of memory locations leads to the corruption of the operating system and the BASIC code.

Related instructions: SEGMENT, PEEK, VARPTR

Graphics instruction

Draws a Train of Lines

Purpose: Draws a train of lines the end points of which have been fixed in an integer field. The field contains relative x,y coordinates (Δ values).

Syntax:

```
[n] POLYLINE a,v%(b)
```

n: line number, also including label
a: number of lines to be drawn
v%: integer field with Δx*4, Δy*4 [,Δxi*4, Δyi*4]...
b: first field element to be used

Remarks: This instruction draws so fast because the coordinates are not converted with the VIEWPORT and WINDOW parameters. The field must therefore contain values which comply with the absolute screen coordinates; for the x-axis, i.e. from 0 (or -80 * 4 = -320) to 399 * 4 = 1596 and for the y-axis from 0 to 639 * 4 = 2556. The graphics cursor for example uses a preceding MOVE command to specify the starting point of the trace which allows to shift the trace easily over the screen. The first line is drawn using the first pair of Δx/ Δy, the second with the next pair and so on until the number a is reached.

Possible error message: *Error 20: "redimensioned array"*

Reason: The number of available field elements must at least be twice the number of lines. E.g., if a = 10 DIM must be v% (at least 19) if drawing is to start from element 0 (b = 0).

Instruction

Character Output on the Screen

Purpose: Transfers data to the screen, in particular character strings and variable lists or expressions.

The PRINT instruction is also used for setting the cursor, setting the attributes and deleting lines and the screen. These functions are controlled using the ANSI functions described in the Section "Screen Function".

Syntax: `[n] PRINT [list of expressions] [;] or [,]`

n: line number, also including label

Remarks: Numeric or string variables can be written directly after the PRINT statement. Several expressions must be separated by semi-colons or commas.

A carriage return and line feed (CR + LF) are automatically generated after each PRINT statement. The next PRINT statement starts at the beginnning of a line. A PRINT statement without any subsequent expression thus generates a blank line.

" can also be used instead of PRINT. The instruction LIST, however, outputs PRINT rather than "".

Example:

```
100 PRINT
200 PRINT A
300 PRINT A$
```

,

Every twentieth print position is internally pretabulated. A comma after the numeric or string expression causes the next character to be written in the next tabulated position. Carriage return and line feed are suppressed.

```
100 PRINT "FREQ.        IN", A$,A
```

;

Carriage return and line feed will also be suppressed if a semi-colon follows the expression. The numbers or strings are linked together.

E_c

Furthermore, the screen attributes are set by means of the PRINT instruction and ANSI sequences. The available sequences are described in the respective device-specific manual.

Example: Output of a remark as a flashing display:

```
100 PRINT "Ec[5m S1 OFF"
110 PRINT "Ec[0m": REM RESET ATTRIBUTS
```

Example: Clearing the screen:

```
100 PRINT "Ec[2J"
```

Related instructions: PRINT#

Instruction

Formatted Output

Purpose: Outputs numbers in a format specified by a character string.

Syntax:

```
PRINT USING s$, [or ;] list
PRINT#a, USING s$, [or ;] list
```

Remarks: Each element of the list is output formatted with the string s$. The string comprises a sequence of dummy values which define the format of a number.

Possible string elements:

- `#` Dummy value for a digit. A right overflow is rounded off; a left overflow causes output of the unformatted number.
  Unoccupied positions before the decimal point result in spaces and after the decimal point in 0.
- `.` Dummy value for decimal point.
- `*` Must be present before the first #. Unoccupied positions before the decimal point are replaced by * (e.g. for filling out spaces before the amount: ***100.- DM).
- `+` Dummy value for the sign. Can be positioned before or after the number. Positive value results in + Negative value results in -.
- `-` Dummy value for the sign. Can be positioned before or after the number. Positive value results in space. Negative value results in -.
- `↑` Dummy value for the exponent. Must be present after the last #.

Example:

```
10 PRINT A,B,C,D
 3.14159265359      -199.99999999          1.234E-4        7.89E+17
10 PRINT USING "###.#",A,B,C,D
  3.1               200.0              0.0             7.89E+17
10 PRINT USING "*######.##",A,B,C,D
****3.14            **200.00           ****0.00        7.89E+17
10 PRINT USING "+####.##",A,B,C,D
+   3.14            - 200.00             + 0.00        7.89E+17
10 PRINT USING "######.##-",A,B,C,D
    3.14              200.00-              0.00        7.89E+17
10 PRINT USING "+   .##↑",A,B,C,D
+3141.59E-3         -2000.00E-1         +1234.00E-7    +7890.00E14
```

Related function: STR$

Instruction

Writing to a File or Interface

Purpose: Once the file has been opened (see OPENO), PRINT# instructions can be used to transfer data to the selected file on the floppy disk or fixed disk via the interface. Several items of data can be present in one PRINT instruction, each separated by a semicolon or a comma.

Syntax:

```
[n] PRINT# a,[list of expressions][;] or [,]
[n] PRINT# a, USING s$, [list of expressions] [;] or [,]
```

n: line number, also including label
a: channel number (1 to 15)

Remarks: Using this instruction, all numeric and string expressions are serially written into the file.

No separators are transmitted between the expressions separated by a semicolon. LF and CR are transmitted at the end of the instruction. These can also be suppressed by a semicolon at the end of the instruction. If separators are required between the expressions, they can be set between quotation marks, e.g. PRINT A; ",";B)

If the data are then to be read using an INPUT# instruction, it must be remembered that not more than 80 characters may be written before the delimiter CR LF.

Related instructions: OPENO#, CLOSE#

Example:

```
90 OPENO# 1,"DATA.ASC"
100 PRINT# 1,A$
110 CLOSE# 1
```

Output of several data sets

```
200 PRINT# 1,A$;16*(X↑2);"KHZ"
```

Output of other delimiter

```
300 PRINT# 1,A$;CHR$(32);
```

Possible error message: *ERROR 55: "file not open for output"*

Reason: The file to be written to has not been opened.

Graphics instruction

Drawing of Lines with Relative Coordinates

Purpose: This instruction acts in the same manner as the DRAW instruction, except that the coordinates x and y refer to the current position of the cursor (Δ values) and not to zero.

Syntax: `[n] RDRAW x,y`

n: line number, also including label
x,y: numeric expressions for the x/y coordinates of the destination point

Related instructions: DRAW, MOVE, WIDTH, SET

Instruction

Read Data

Purpose: The characters of the DATA lines are read in up to the first comma by means of the READ statement and assigned to the numeric or string variable.

Syntax: `[n] READ v1 [,vn]`

n: line number, also including label
v: numeric or string variable

Remarks: Several variables can be read at a time with the READ statement. The READ statements may be present at different places in the program. A pointer indicates the last DATA character read and reading is recommenced at this position with the next READ statement.

Related instructions: `DATA, RESTORE`

Example:

```
100 DATA 123,456
110 DATA 789
130 READ A : PRINT A
```

Output: *123*

Possible error message: *ERROR 39: "out of data"*

Reason: An attempt has been made to execute more READ statements than data available.

This page has been kept free on purpose. The BASIC instruction of an option may be inserted at this place. The pages to be inserted are found in the manual of the respective option.

Instruction

Remark

Purpose: It will be useful to provide a program with comments or with a heading if complex problems have to be solved using a large number of subroutines. The REMARK instruction is used for this purpose.

Syntax:

```
[n] REM any character sequence
[]: ' any character sequence
```

n: line number, also including label

Remarks: With the REM statement, all subsequent characters are ignored and the next line is then processed. The characters do not influence the program run, but appear in the listing and thus facilitate programming. Programmers with practice use the instruction for program debugging in locating unwanted statements. The instructions are retained, but are not processed by the controller. The section of the program can be incorporated again by deleting the REM statement. The REM statement together with the subsequent characters must not be longer than 80 characters. The apostrophe character is equivalent to REM.

Note: ':' is part of the remark and does not separate instructions.

Example:

```
100 REM **** SUBROUTINE 1 ****
120 ' Evaluation of measurement
```

Command

Renumbering of Lines

Purpose: Further lines must often be inserted into an already existing program. This will only be possible, however, if the interval between line numbers is greater than 1. If this is no longer the case, larger intervals can be generated again by renumbering using the instruction here described. (The jump addresses with the instructions THEN, GOTO, GOSUB, and RESTORE are also modified according to the new numbering.)

Syntax: `RENUMBER [m1] [-[m2]] [,n[,Δn]]`

n: new first line number
Δn: increment
m_1: old line number at which renumbering is started
m_2: old line number up to which renumbering is carried out

Default values: n = 10
Δn = 10

Remarks: An area (from m_1 to m_2) within a program can be renumbered using this instruction. The sequence of lines cannot be changed with this instruction! Blocks can, however, be combined in any sequence by renumbering and loading them onto a floppy disk (ASAVE, ALOAD).

Examples: Renumber the complete program to initial line 10 and step size 10.

```
RENUMBER
```

The program part from line 25 to line 333 inclusive is renumbered to initial line 10 and step size 10.

```
RENUMBER 25-333
```

The complete program is assigned the step size 20 and the initial line 100.

```
RENUMBER, 100, 20
```

The program part from line 25 to line 333 inclusive is assigned the new initial line 100 and the step size 20.

```
RENUMBER 25-333, 100, 20
```

Possible error message: *"Lines nested"*

Reason: The new line number range would cover up already existing line numbers.

Instruction

Loop Instruction/Structure Element

Purpose: Defines a loop with any number of lines (instructions) which will be repeated until the condition specified at the end is true.

Syntax:

```
[n] REPEAT
[n]   instruction
          .
          .
          .
[n]   instruction
[n] UNTIL a
```

n: line numbers, also including labels
a: expression

Remarks: This loop is executed once in any case (contrary to WHILE-WEND). The expression is calculated at the end of the loop and, if the result is false (i.e. 0), the program will again branch to the line with the associated REPEAT instruction. If the result is true, the program will be continued with the instruction following UNTIL.

Nestings, even with different structure elements, are permissible. In this case, the inner loop instructions should be indented to ensure better readability. GOTOs into loops and out of loops should be avoided, if possible.

Related instructions: WHILE-WEND, FOR-NEXT

Example:

```
10 Min=3: Max=20
20 REPEAT
30   INPUT "INPUT(3...VALUE...20):",VALUE
40 UNTIL Value>=Min AND <=Max
```

```
10 Factor=1
20 REPEAT
30   PRINT Factor
35   Factor=Factor*1.1
40 UNTIL Factor>=10
```

Sorting program (Bubble-Sort)

```
10 REPEAT
20   Flips=0
30   FOR I=1 TO J-1
40    IF A$(I)>A$(I+1) THEN
50     B$=A$(I): A$(I)=A$(I+1): A$(I+1)=B$:REM SWAP
60     Flips=1
70    ENDIF
80   NEXT I
90 UNTIL Flips=0
```

Command

Replacing Text Passages

Purpose: The text passage to be replaced is first searched for in the line range given. When found, the line is output on the screen and the new text is substituted for the text passage to be replaced. After pressing of the return key the new line is returned to the program.

Syntax: `REPLACE [n-m,] t1, t2`

n-m: range of lines in which the search is conducted. If it is not entered, the search will be executed in the complete program.
t1: text searched for to be replaced
t2: new text

Remarks: If any other key than the return key is pressed following this command, it will possible to change the line with the screen editor as is also the case with the SEARCH command. The line changed is returned to the program using the return key and the next line is searched for in the range given and then output.

The text t1 to be replaced must be written exactly as it appears in the listing, i.e. keywords must be written in upper-case letters and the first letter of variable names must be capitalised. Also the blank spaces are important. As to the replacing text, the notation is arbitrary. BASIC again verifies the correctness of the entire line with regard to mistakes in syntax, as is also done when a new line is entered.

Related commands: `PLIST, COPYOUT, OPENO LPT`

Example:
```
REPLACE Cn, Count
REPLACE 100-200, Cn, Count
```
replaces the variable name Cn by Count in the line range from 100 to 200 inclusive.

```
REPLACE 1000-2000, 3.14, PI
REPLACE, 3.14, PI
```
replaces the figure 3.14 by the variable PI in the entire program.

Note: The comma is separator and cannot be part of the text to be replaced.

Instruction

Restore Data Pointer

Purpose: RESTORE can be used if data are to be read from the beginning with further READ lines or from particular DATA lines.

Syntax: `[n] RESTORE [m]`

n: line number, also including label
m: line number to which data pointer is set

Remarks: If RESTORE contains no line numbers, the pointer will be set again before the first character of the first DATA line and with the next READ the first data are read.

With a specified line number given in the RESTORE instruction, the data pointer is set to the first character of the DATA line following the given line.

If a line number has been specified with the RESTORE instruction, the data pointer is set to the first character of the DATA line indicated, or, if not existent, to the next following DATA line.

Related instructions: READ, DATA

Example: Set data pointer to line 200

```
100 DATA 1, 2, 3, 4, 8, 255
200 DATA TEXT, PRO6, START, DEF
210 READ A,B
220 RESTORE 200
230 READ A$

A$ = "TEXT"
```

Note: RUN automatically executes the RESTORE instruction.

Instruction

Return from the Subroutine

Purpose: This instruction indicates the end of a subroutine. The program is continued with the next line following the GOSUB call.

Syntax: `[n] RETURN [a]`

n: line number, also including label
a: level

Remarks: The parameter a optionally specified indicates the level to be returned to. If no parameter is indicated, the program will be continued with the instruction following the calling GOSUB. With positive numbers, level a is selected with 0 as the highest level - in general the main program - and the level is incremented following each GOSUB. The following example branches back into the main program, e.g. following an error handling.

```
100 RETURN 0
```

With negative numbers, a levels are skipped. RETURN -1 is thus equivalent to RETURN without parameter. RETURN -2 returns to the last but one GOSUB etc.

Related instructions: GOSUB, ON GOSUB

Example:

```
100 A=5:B=1:GOSUB1000:PRINT"RESULT",U
110 END
1000 REM SUBROUTINE
1100 U=(A+B)*3: RETURN
```

Possible error message: *ERROR 34: "RETURN without GOSUB"*

Reason: RETURN has been reached without a previous GOSUB, e.g. if 110 END were left out in the example above.

String function

Separate Last Character from a String

Purpose: Similarly to the LEFT$ function, this function is used to read a number of characters specified in this function and process characters from the right-hand end of a string, the start of the string being retained.

Syntax: `RIGHT$(s$,a)`

s$: string variable or string constant
a: number of characters

Related instructions: `LEFT$, MID$`

Example:

```
100 A$="PROCESSCONTROLLER":PRINT RIGHT$(A$,6)
```

Display: *ROLLER*

Graphics instruction

Moving the Graphic Cursor

Purpose: This instruction acts in the same manner as the MOVE instruction except that the coordinates x and y refer to the current position of the (imagined) pencil (Δ-values) and not to zero.

Syntax: `[n] RMOVE x,y`

n: line number, also including label
x,y: numeric expressions for the x/y relocations

Related instructions: MOVE, DRAW, RDRAW, LABEL, AREA

Numeric function

Random Function

Purpose: BASIC is provided with a random number generator which generates pseudo-random numbers between 0 and 1. The generator is called up using RND (random).

Syntax: `RND(a)`

a: constant, variable or numeric expression

Remarks: 3 different modes can be used:

1) a negative

RND with a negative argument always assigns the same random number to each a. The random numbers can then be reproduced, which may be useful for program development or troubleshooting.

2) a = 0

The RND function repeats the last value.

3) a positive

With a positive argument, the RND function provides the next number in sequence.

Example:

```
100 Y=RND(X)
```

Electronic dice

```
100 W=INT(6*RND(1))+1:PRINT W
```

In order to obtain a changing random number following RUN (randomize), the following line is used:

```
10 A=RND(-VAL(RIGHT$(TIME$,2))/1000)
```

Command

Program Start

Purpose: The RUN instruction is used to start a program already in store. The controller then executes the program starting with the first line.

Syntax:

```
RUN
```

Remarks: The following is used if the program is to be started at a higher line and not at the beginning:

Syntax:

```
RUN m
```

m: line number to be started at

Remarks: All variables, field variables etc. in the memory are deleted when the RUN command is executed so that the controller is present in the same status at the beginning of the program as that following loading of the BASIC interpreter. All the basic statuses described in the respective instructions are set. In particular, files which may still be open are closed (CLOSE).

Related command:

```
CONT
```

Example:

```
RUN

RUN1200
```

Note: RUN can also be entered by pressing the softkey 2.

Instruction

Saving Programs

Syntax: `[n] SAVE s$ [,P]`

n: line number, also including label
s$: program name, also with drive and search path
P: character P or p saves program ('protected' storage)

Remarks: The program present in the main memory of the computer is saved on floppy disk or fixed disk following this instruction.

Max. 8 characters and one extension separated by a dot may be entered as program name, i.e. the file name must be in accordance with the MS-DOS format. If no extension is used (i.e. a name without dot), BASIC will automatically use the extension ".BAS". If the program name is to remain without an extension, "." must be written as last character.

Following loading a program protected by the parameter 'P' can neither be looked into with LIST or PLIST, nor be stored in ASCII form with ASAVE. Attempts to do so cause the error message:

'protected file loaded!'.

A protected program can load further programs with CHAIN and vice versa, also protected programs can be reloaded with CHAIN, but in those cases the whole program is handled as protected program.

Related instructions: LOAD, ASAVE

Example: 1. Saving the BASIC program on the default drive

```
SAVE "TEST.BAS"
```

2. Saving on the fixed disk drive

```
SAVE "E:TEST.BAS"
SAVE "E\USER\TEST.BAS"
```

3. Protected saving of program:

```
SAVE "PROT.BAS",P
```

Note: Contrary to the ASAVE command, the SAVE instruction stores the interpreter code of the BASIC on floppy disk or fixed disk. This is shorter than the LIST file and need not first be translated when loading.

Instruction

Increase in the BASIC Run Speed

Purpose: Increasing the BASIC run speed in the first run.

Syntax: `[n] SCAN`

n: line number

Remarks: So as to approximate the speed of the BASIC interpreter to the speed of a compiler the jump destinations of GOTO/GOSUB instructions are filled in tables so that they need to be searched for only once in the user program. Thus, the first run of the program following RUN lasts slightly longer, because the table is only gradually completed. In case of highly time-critical programs when the program has to be executed also in its first run at maximum speed, the completion of the table of jump destinations may be forced by means of the SCAN-command before starting "RUN".

Particularly long programs having many jump branches require a search in the program of several seconds.

Graphics Instruction

Setting the display mode

Purpose: The graphics adapters for the PSA/PAT controllers support different operating modes. The respective operating mode also depends on the kind of monitor connected. After power on the hardware is checked in the following order and SCREEN set to the optimum value: 18 (VGA), 17 (VGA S/W), 16 (EGA), 15 (EGA SW), 8 (Herc.) and 6 (CGA). This instruction is used to switch over to another mode for test purposes.

Syntax: `[n] SCREEN a`

n: line number , also including label
a: expression for the mode according to the description below

Remarks:

SCREEN 3

Alphanumeric mode defined by IBM, which is available on each hardware (CGA, EGA, VGA and MCGA) and compatible to the software.

Alphanumeric resolution: 25 lines, 80 characters per line

Attributes: 8 foreground and 8 background colors (only in case of graphics color monitors), foreground brightness, blinking.

1 status line (and 3 softkey lines); scrolling range 24 (21) lines.

SCREEN 4

CGA (color graphic adapter) mode defined by IBM. This mode is also emulated by the EGA and VGA hardware.

3 colors + background of 2 selectable palettes.

Alphanumeric resolution: 25 lines, 40 characters per line.

1 status line (and 3 softkey lines); scrolling range 24 (21) lines.

Graphics resolution: X-axis 320 pixels, Y-axis 200 pixels.

Colors: 2 palettes with 3 colors and one background color (black) each

Special features: The color code is stored with 2 bits per pixel. The information about color graphics contained in the description of the SET and COLOR instruction is not relevant for this mode; there are no 4 planes per pixel.

Graphics Instruction

COLOR x,N,x,x selects palette 1 of N using even numbers and palette 2 using odd numbers. The x-parameters are variables.

SET a,N selects the follwing colors:

N	palette 1	palette 2
0	black (background)	black (background)
1	violet	yellow
2	cobalt blue	red
3	white	green

LABEL a$,N creates the same character size with N = 0 and N = 1 , since the resolution in the x-direction is not sufficient for size 0.

SCREEN 6

CGA black/white mode, which is also emulated by the EGA and VGA hardware.

Colors: black/white

Alphanumeric resolution: 25 lines, 80 characters per line.

1 status line (and 3 softkey lines); scrolling range 24 (21) lines.

Graphics resolution: X-axis, 620 pixels, Y-axis 200 pixels.

SCREEN 7

Alphanumeric mode defined by Hercules, which is emulated by the multifunction board.

Alphanumeric resolution: 25 lines, 80 characters per line.

Attributes: foreground brightness, blinking, underlining.

1 status line (and 3 softkey lines); scrolling range 24 (21) lines.

Special features:

The Hercules board and compatible boards directly support this mode. A few other multifuntion boards must first be prepared by means of the HERCMOD.COM program, e.g. by adding \basdrv\hercmod to the AUTOEXEC.BAT file.

Graphics Instruction

SCREEN 8

Graphics mode defined by HERCULES, which is emulated by the multifunction board.

Colors: black/white

Alphanumeric resolution: 25 lines , 80 characters per line.

1 status line (and 3 softkey lines); scrolling range 24 (21) lines.

Graphics resolution: X-axis 720 pixels, Y-axis 348 pixels.

Special features: The Hercules board or compatible boards support this mode. A few other multifunction boards must first be prepared by means of the HERCMOD.COM program, e.g. by adding \basdrv\hercmod. to the AUTOEXEC.BAT file.

For alphanumeric outputs in graphics mode the support routines must be resident. This is achieved by means of the program HERCSUP.COM, e.g. by adding \basdrv\hercsup to the AUTOEXEC.BAT file.

SCREEN 14

EGA (Enhanced Graphics Adapter) mode defined by IBM for low resolution displays (15.75 kHz), which is also emulated by the VGA hardware.

Colors: 16 of a 64-color palette.

Alphanumeric resolution: 25 lines, 80 characters per line.

1 status line (and 3 softkey lines); scrolling range 24 (21) lines.

Graphics resolution: X-axis, 640 pixels, Y-axis 200 pixels.

Special features: COLOR f, r, g, b offers only 4 saturations for each of the 3 basic colors. They are assigned as follows for reasons of compatibility:

r, g, b	color saturation
0,1	lowest
2,3	
4,5,6,7	
8 to 16	highest

Graphics Instruction

SCREEN 16

EGA graphics mode defined by IBM for medium resolution displays (21,85 kHz), which is also emulated by the VGA hardware.

Colors: 16 of a 64-color palette

Alphanumeric resolution: 25 lines, 80 characters per line.

1 status line and 3 softkey lines; scrolling range 24 (21) lines.

Graphics resolution: X-axis 640 pixels, Y-axis 350 pixels.

Special features: COLOR f, r, g, b offers only 4 saturations for each of the three basic colors r,g,b. They are assigned as follows for reasons of compatibility:

r, g, b	saturation
0,1	lowest
2,3	
4,5,6,7	
8 to 16	highest

SCREEN 17

MCGA (Multicolor Graphics Array) and VGA (Video Graphics Array) mode for the high resolution monochrome monitor (31,5 kHz). This mode is compatible to a large extent to the PCA controller graphics without color option.

Alphanumeric resolution: 30 lines 80 characters per line.

5 status and softkey lines; scrolling range 25 lines.

Graphics resolution: X-axis 640 pixels, Y-axis 480 pixels.

SCREEN 18

VGA mode defined by IBM for the high resolution monitor (31,5 kHz). This mode is to a large extent compatible to the PCA controller graphics with color option PCA-B3.

Colors: 16 of a 256000-color palette

Alphanumeric resolution: 30 lines 80 characters. per line.

Graphics Instruction

5 status and softkey lines; scrolling range 25 lines.

Graphics resolution: X-axis 640 pixels, Y-axis 480 pixels.

Special features: COLOR f, r, g, b offers 64 intensitiy degrees for each of the three basic colors r,g,b;the range of numbers defining the color portion is thus increased to 64.

SCREEN 19

VGA or MCGA mode defined by IBM for the high resolution monitor (32.5 kHz) with low resolution graphics but extended color options.

Colors: 256 of a 256000 color palette.

Alphanumeric resolution: 25 lines, 80 characters per line.

1 status line (and3 softkey lines); scrolling range 24 (21) lines.

Graphics resolution: X-axis 320 pixels, Y-axis 200 pixels.

Special features: COLOR f, r, g, b offers 64 saturations for each of the three basic colors r,g,b; the range of numbers defining the color portion is thus increased to 64.

Survey of the modes

SCREEN	mode	pixels	colors/ grey level	palette	alphanum. characters	PSA	PAT
3	alphanum.				80 x 25	+	
4	CGA	320 x 200	4	2	40 x 25	•	
6	CGA	640 x 200	b/w	-	80 x 25	•	
7	Herc				80 x 25		+
8	Herc	720 x 348	b/w	-	80 x 25		+
14	EGA	640 x 200	16	64	80 x 25	•	
16	EGA	640 x 350	16	64	80 x 25	•	
17	MCGA	640 x 480	b/w	-	80 x 30	•	
18	VGA	640 x 480	16	256 000	80 x 30	+	
19	VGA	320 x 200	256	256 000	40 x 25	•	

+ recommended, • possible

Command

Search for Certain Texts

Purpose: The SEARCH command enables all lines of a BASIC program to be output on the screen which contain a certain text. For example, all lines can be listed which contain a PRINT or INPUT statement or a particular variable.

Syntax:

```
[n] SEARCH [n - m,] t
```

n - m: range of lines to be searched The complete program will be searched if this is not entered
t: text searched for

Remarks: A line is output on the screen when found. It is possible to modify this line using the screen editor. The modified or unmodified line is returned to the program using the return key and the next line is searched for in the range specified and then output.

The search text must be written exactly as it appears in the listing, i.e. keywords must be written in upper-case letters and the first letter of variable names must be capitalized, etc. Also the blanks are important.

Related command: REPLACE

Example:

```
SEARCH GOSUB
SEARCH 100-200, GOTO
```

searches for the word "GOTO" between lines 100 and 200.

```
SEARCH 100-200, 888
SEARCH , 888
```

searches the complete program for the number 888

Instruction

Determining the Segment

Purpose: The CPU used in this controller requires information on the address within a segment as well as the segment position in order to determine addresses. The segment is specified using the SEGMENT instructions and a subsequent integer.

Syntax: `[n] SEGMENT a` oder `DEF`

n: line number, also including label
a: segment address (-32768 to 65536)
DEF: keyword for default setting

Related functions: `POKE, PEEK, VARPTR`

Example:

```
10 SEGMENT HEX("12F3")
100 SEGMENT DEF
```

Remarks: Following RUN, the segment is initialised to the value of the BASIC variable memory.

PEEK, POKE, VARPTR and CALL are therefore used to reach memory locations within the 64K variable memory.

The position (the offset) of a variable, which can be determined using the function VARPTR ( ), refers to this initialised segment address.

DEF is a keyword which, together with the SEGMENT instruction, causes the segment address to be reset to the BASIC variable memory. DEF should only be used in association with SEGMENT.

Graphics instruction

Types of Display

Purpose: The SET instruction specifies the type of representation used for drawing with the following instructions. (The INVERT instruction, on the other hand, inverts the screen output).

Syntax: `[n] SET a [,b] [,c]`

n: line number, also including label
a = 0 : blanked characters
a = -1: inverted characters
a = 1 : unblanked characters (default)
a = 2 : dominant color (only with color graphics option)
b: color pen 0 to 15 (or 255*) (default 1): must be set to 1 if no color graphics option is installed
c: 1 = blank graphics, fast drawing (only PCA)
0 = display graphics (default)
2 = switch off built-in screen

*) see SCREEN instruction

Remarks: Blanked characters are drawn using SET. This instruction is used to delete individual lines or points or to draw blanked characters on the unblanked screen.

SET-1 subsequently inverts everything drawn, which means that unblanked dots are blanked and blanked dots are unblanked. Inversion continues until the SET-1 instruction is replaced by a different SET instruction.

Example: The central part of a line is deleted.

```
100 MOVE 20,100
110 DRAW 300,100
120 HOLD 1000
130 SET 0
140 MOVE 110,100
150 DRAW 210,100
```

Example: A line is continuously inverted.

```
100 CLEAR
110 SET -1
120 MOVE 160,20
130 DRAW 160,179
140 HOLD 100
150 GOTO 120
```

SET1 corresponds to the usual setting of the controller, i.e. unblanked characters are drawn on the blanked screen. This instruction is used to reset the normal type of display following SET 0 or -1.

Summarizing, the following applies to the first parameter a:

- SET only refers to the type of display of DOT, DRAW, LABEL, AREA and RDRAW.
- The SET instruction continues to apply in the program until deleted by another SET instruction.
- SET1 is the default setting upon switching on the controller.

The parameter b is used to select the color memory with which all following graphic instructions are to be executed. Only 1 may be specified as the parameter without the option PCA-B3. 16 color values displayed simultaneously (b = 0 to 15) are available with PCA-B3 (see SET using color graphics option).

The graphics can be blanked using the parameter c, e.g. in order to output the ASCII characters (PCA). Graphic instructions can still be used to write into the graphics memory even if the graphics is blanked. This takes place at a higher speed and can therefore be used for a faster production of graphics.

In the case of c having the value 2, the built-in screen is switched off, e.g. to further reduce the RF leakage. The horizontal deflection is switched off and thus also the generation of high voltage. If c = 0 or 1, the screen will be switched on again.

SET Using Color Graphics

(Option PCA-B3 or VGA and EGA mode for PSA and PAT)

The color of every point of the color graphics is determined by four memories (planes). Thus, 16 colors can be displayed simultaneously. The parameter b of the SET instruction selects the pen to be subsequently used for drawing. The color of this pen which will then be displayed on the screen is only determined by the COLOR instruction (4096 possibilities) (see COLOR).

Pens 1, 2, 4, 8 draw particulary fast because only one bit is set in each of these binary numbers and, only one plane needs therefore be written to. Pen 15 is the slowest.

If two color areas drawn with different colors overlap, e.g. the figures drawn with pens 2 and 4, the overlapping area has the same color as pen 6. The area where two colors overlap takes on the color which is produced by logical ORing of the binary numbers of the individual pens.

What has been said so far applies to the non-dominant mode which will be set if the parameter a of the SET instruction has the values -1 (inverted drawing), 0 (reset dots) or 1 (set dots). If the parameter a has the value 2, the dominant mode is cut in. The figure drawn takes exactly the color that corresponds to the pen; colors drawn first disappear. Thus, all four planes must always be used, which is why the speed of drawing decreases just as is the case with pen 15 in the non-dominant mode.

Related instructions: COLOR, WIDTH, DRAW, RDRAW, AREA, LABEL, DOT, SCREEN

Numeric function

Sign Function

Purpose: If only the sign of a number or function is of interest, it will be gained using the SGN function.

Syntax: `SGN(a)`

a: numeric expression

Remarks: The SGN function provides the following results:

SGN(a) = 1 for a > 0
= 0 for a = 0
= -1 for a < 0

Example:

```
100 B=SGN (A)
```

Instruction

Operating System Call

Purpose: Loads and starts other MS-DOS programs (e.g. with the .COM, .BAT or .EXE extension). After completion of these programs, BASIC continues with the next instruction after SHELL. SHELL entered in direct mode permits access to the command interpreter of MS-DOS via the keyboard.

Syntax:

```
[n] SHELL [s$]
```

n: line number, also including label
s$: string transferred to the command interpreter of MS-DOS, optionally with further parameters for the program to be called.

Remarks: Using the SHELL instruction without string as parameter, the command interpreter is loaded from the mass storage and called up. MS-DOS registers e.g. with "E>". Then all MS-DOS programs such as DIR, COPY, DEL, CHDIR, PRINT, etc. can be called. To return to BASIC, enter the word EXIT. Command files can also be started in this manner. In this case, EXIT must be the last instruction of this file.

Related instructions: BYE, DIR

Example:

```
SHELL
A>DIR                    (MS-DOS waits for a command to be entered; in this case, user types 'DIR')

A>EXIT                   (user wants to return to BASIC).
```

The same result is obtained by means of the following entry:

```
SHELL "DIR"
SHELL "DIR *.BAS"        (provides all files with the extension .BAS)
```

Example: The following example generates a file, e.g. with measured values, calls the sort program and then processes the sorted values, e.g. for graphic output.

```
100 OPENO#1, "MESSIN.DAT"
110 REM Writing the data
.
.
.
200 CLOSE#1
210 SHELL "SORT <MESSIN.DAT >MESSOUT.DAT"
220 OPENI#1, "MESSOUT.DAT"
230 REM Reading the sorted data
```

Note: This instruction is used to load the command interpreter of the operating system from the mass memory into the main memory. The command interpreter must therefore be available in the default directory, for otherwise MS-DOS will respond with "general failure". COMMAND.COM can be loaded from a different directory by means of

SET COMSPEC = E:\COMMAND.COM

in the file AUTOEXEC.BAT. In this example, BASIC will attempt to load the command interpreter from the root directory of drive E:.

If the available memory location for loading the COMMAND.COM file is no longer free (e.g. in case of a too small memory capacity or a very large virtual drive), MS-DOS will signal "generale failure".

Numeric function

Sine Function

Purpose: To generate the sine value of the argument in radian measure.

Syntax: `SIN(a)`

a: constant, variable or numeric expression

Related functions: `COS, TAN, ATN, PI`

Example:

```
100 C=SIN(A)
```

An argument specified in degrees can be modified by conversion.

```
100 C=SIN(A*PI/180)
```

If the argument is specified in centesimal degrees, the conversion will be as follows:

```
100 C=SIN(A*PI/200)
```

Instruction

Restoring the Softkey Labelling

Purpose: If the user has deleted the softkey labelling, e.g. because he labels the softkeys according to his special needs, the softkey labelling as it appeared in the two bottom lines after calling BASIC can be restored using this instruction.

Syntax: `[n] SOFTKEY`

n: line number, also including label

Example:

```
100 PRINT "Ec[y":REM       Deletion of softkey labelling
110 PRINT "EcR1 Ja ":REM    new labelling
200 SOFTKEY           :REM  Restoring the softkey labelling
```

(See section 1.3.3)

Numeric function

Square Root Function

Purpose: The square root of the argument is generated using SQR.

Math.:

$$y = \sqrt{x}$$

Syntax: `SQR(a)`

a: constant, variable or numeric expression

Example:

```
100 Y=SQR(X)
```

Possible error message: *ERROR 31: "numeric overflow"*

Reason: Negative argument a

Instruction

Stop Statement

Purpose: The STOP statement can be used e.g. when debugging programs in order to stop the program at a critical point and to read variables. The program can subsequently be continued using the CONT command.

Syntax: `[n] STOP`

n: line number, also including label

Remarks: The STOP statement also differs from END in that the line number in which the program has been interrupted is also output.

Related instructions: `END, TRACE, CONT`

Example:

```
100 STOP
```

Output on screen: `STOP IN LINE 100`

String function

Conversion of Numeric Variables into String Variables

Purpose: Contrary to the VAL function, also numeric expressions can be converted into string variables, e.g. to insert numbers into string variables.

Syntax: `STR$(a[,USING s$])`

a: constant, variable or numeric expression
s$: String comprising dummy values for formatted output (see PRINT USING)

Related functions: `VAL, PRINT USING`

Example:

```
50 A$=STR$(B)
100 A$=STR$(B+12.5)
120 A$=STR$ (B,USING"##.##")
200 IECOUT3,STR$(A1,USING"###")
```

Remarks: STR$ with USING must not be part of a string linkage. The following is therefore not permissible:

A$ = "FR" + STR$ (B, USING "##.##")

When the formatted number does not match with the format, no error message is output but "USING" is ignored.

Function

Distance from Left Edge of Screen

Purpose: The TAB function of the PRINT statement serves the purpose of printing characters at a preset distance from the left edge of the screen.

Syntax: `TAB(a)`

a: constant, variable or numeric expression (0 to 79)

Remarks: The parameter a can be a constant, variable or expression with decimal places being left out. If the write position assigned by the TAB instruction is already occupied by a previous PRINT statement, printing is carried out at the next free position. The TAB instruction must be separated from the other expression by a semicolon.

Example:

```
100 PRINT "MEASURED VALUE";A;TAB(17);"IN MS"
```

Display if A = 1.23456

MEASURED VALUE 1.23456 IN MS

Note: BASIC cannot read the current interface position. If, e.g., the cursor position was changed via the Escape sequences or characters output, which cannot be printed, BASIC cannot reach the write option desired. BASIC is not informed until the next CR that the internal character counter must be reset.

Numeric function

Tangent function

Purpose: Transfers the tangent of a.

Syntax: `TAN(a)`

a: constant, variable or numeric expression

Remarks: The cotangent function is not available in the instruction set of the controller but may be generated by a simple conversion.

Math.:

$$cot(x) = \frac{1}{tan(x)}$$

Related functions: `ATN, COS, SIN`

Example:

```
100 Y=1/TAN(X)
```

Pseudo variable

Time Measurement

Purpose: The pseudo variable TIME is available for time measurements and for calculating the time at high resolution. The time, starting at midnight, is increased by 1 every 10 ms.

Syntax: `TIME`

Remarks: The relative time is entered by assigning the pseudo variable TIME to a numeric variable. The system units are hundredths of a second (10 ms).

If the option real-time clock is provided, the time is read out from the option.

The clock is set at the operating system level.

Related Pseudo variable: `TIME$, DATE$, DATUM$`

Example: Measuring of time difference:

```
100 A=TIME
110 HOLD 1000
120 B=TIME
130 PRINT B-A
```

Note: The maximum value of TIME may amount to:

100•60•60•24 = 8 640 000

Pseudo variable

Readout of Time

Purpose: The absolute time is read out using the pseudo variable TIME$.

Syntax: `TIME$`

Remarks: The time is transferred as a string with 11 characters in the form

```
hh:mm:ss.tt
```

hh = hour
mm = minute
ss = second
tt = hundredth of a second

If the option real-time clock is provided, the time is read out from the option. The clock is set at the operating system level.

Related Pseudo variable: `TIME, DATE$, DATUM$`

Example:

```
100 A$ = TIME$
```

Instruction

Program Tracing

Purpose: The number of each line of a running program is output on the screen before the line is executed.

Syntax: `[n] TRACE`

n: line number, also including label

Purpose: If TRACE is followed by at least one expression or e.g. a list of variables, TRACE will change into step mode. Following the execution of each line, the line that has just been executed and the value of the expressions (or variables) is output on the lower edge of the screen. The controller waits for further entries; e.g. when CONT is entered, the program continues until the next line is reached.

Syntax: `[n] TRACE a/s$ [,a/s$]...`

Example:
```
TRACE A,B/256, A$(N,M)
```

Purpose: The output of the line numbers of the running program activated by the TRACE instruction is switched off again by the TRACE OFF instruction.

Syntax: `[n] TRACE OFF`

Note: The instruction TRACE [] entered in direct mode is maintained after all commands (like RUN) and must be switched off explicitly using TRACE OFF.

Instruction

TTL I/O Interface Access

Purpose: This instruction is used to address the option PCA-B11.

Syntax:

```
[n] TTL z IN  s$,v
    TTL z OUT s$,b
```

n: line number, also including label
z: number of interface (1 to 3)
s$: string expression for setting the interface
v: numeric variable into which the value is read (typical integer variable)
b: numeric expression for the integer value read

Remarks: The detailed description of these instructions is part of the PCA-B11 manual. It also contains pages which may be inserted in this part of in the BASIC manual.

This page has been kept free on purpose. The BASIC instruction of an option may be inserted at this place. The pages to be inserted are found in the manual of the option concerned.

Instruction

Loop Instruction (End)

See REPEAT instruction.

Numeric function

Conversion of Strings into Numerical Values

Purpose: Strings or texts cannot be handled mathematically. Numerical values are, however, often read into string variables via interfaces. The VAL function serves for the conversion into a numerical value on which mathematical operations can be performed.

Syntax: `VAL(s$)`

s$: string expression

Remarks: Leading figures, signs, decimal point and the exponent E are all taken into account until the first non-numerical character and then processed.

If letters or other characters appear at the first position, these are skipped until the first character, that can be used, appears.

Related function: STR$

Example:

```
100 B$="-1.5E2 DM": PRINT VAL(B$)
```

Display: *-150*

```
200 C$="AG+1.248E19": PRINT VAL(C$)
```

Display: *1.248E+19*

```
100 C$="AG+1.248E19TEST1": PRINT VAL(C$)
```

Display: *1.248E+19*

Function

Reading in the Variable Pointer

Purpose Using the function VARPTR, the pointer can be read into the variable specified in v and the position within the 64 Kbyte BASIC data segment at which the variable is filed can be found.

Syntax: `VARPTR (v)`

Remarks: VARPTR specifies the address in the form of an offset. The first address of a variable or the string identifier is output. The offset will be valid only if no segment instruction has preceded or if the previous setting is reset with SEGMENT DEF.

Related instructions: `SEGMENT, PEEK, POKE`

Example:

```
10 DIM A%(100) 20 PL=VARPTR (A% (0))
```

The lowest address of field A% is determined with line 20.

```
10 A$="ABCDEFG"
20 A=VARPTR(A$)
30 N=PEEK(A)+256*PEEK(A+1): REM   NUMBER
35 Ad=PEEK(A+2)+256*PEEK(A+3): REM   ADDRESS
40 FOR I=Ad TO Ad+N-1: PRINT PEEK(I);: NEXT
100 PRINT
READY
RUN

65 66 67 68 69 70 71
```

(see section 1.4.4.1)

Relative Screen Coordinates, Display Area of Screen

Purpose: If the coordinate system selected using the WINDOW instruction is to only fill a part of the screen, the VIEWPORT instruction is used. It also determines the position on the screen and the dimensions of the coordinate system:

Syntax: `[n] VIEWPORT x1,x2,y1,y2`

n: line number, also including label
x1...y2: numeric expressions for the display limits of the screen
x1: left-hand limit
x2: right-hand limit 0 to 639
y1: lower limit y2
y2: upper limit -80 to 399

(-1238 is the lowest limit in the graphics memory outside the screen).

Remarks: The range of the absolute screen coordinates (X = 0 to 639 and Y = -80 to 399) always applies to x1 to y2 independent of whether other values have been set using the WINDOW instruction.

Fig. 2-2 shows the position of the display area using the VIEWPORT instruction.

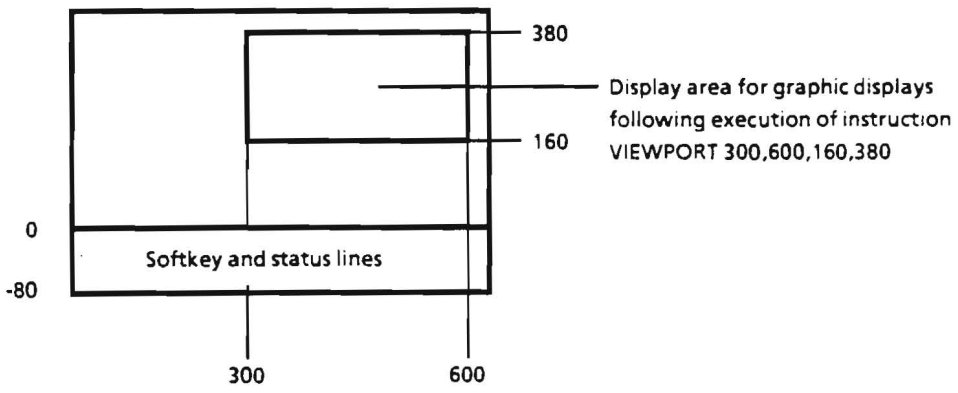


Fig. 2-2 Display area of screen

Remarks: After starting BASIC with RUN, VIEWPORT 0, 639, 0, 399 is initialized. Thus, the lower limit of the coordinate system defined by WINDOW is above the 5 softkey and status lines. If the whole screen is to be used for graphics output, this will be achieved using VIEWPORT 0,639,-80,399. The 0 point of the y axis is then no longer above the status lines (with WINDOW A,B,O,D), but at the lower edge of the screen. Selecting WINDOW 0, 639, 0,479 again produces a coordinate system where each dot is derived from an integer.

Purpose: Graphic displays with different scales or the simultaneous display of several functions or curves at different positions on the screen can be carried out easily using VIEWPORT.

Related instructions: WINDOW, ZOOM

Example: Drawing of two circles at different places of the screen

```
100 WINDOW -1,1,-1,1
110 CLEAR
120 VIEWPORT 0,199,0,199
130 GOSUB Circle
140 VIEWPORT 200,399,200,399
150 GOSUB Circle
160 END
170CIRCLE:
175   MOVE 1,0
180   FOR Winkel=0 TO PI *2 STEP PI /20
190     DRAW COS(Winkel) ,SIN(Winkel)
200   NEXT
210 RETURN
```

Note: Both instructions WINDOW and VIEWPORT only have an effect on graphic displays that are generated in the subsequent program run. Displays stored in the video memory are no longer changed.

Instruction

Loop instruction (end)

See WHILE instruction

Instruction

Loop instruction/structure element

Purpose

Defines a loop with any number of lines (instructions) which are executed as long as the condition specified at the beginning is fulfilled (possibly never).

Syntax:

```
[n] WHILE a
[n]    instruction
.
.
.
[n] WEND
```

n: line numbers, also including labels
a: expression

Remarks:

The expression after WHILE is calculated and, if true (i.e. not 0), the instruction following WHILE is executed. Otherwise, the associated WEND instruction is searched for and the following program executed.

Nesting, even with other structure elements, is permitted. The inner loop instructions should then be indented to ensure better readability. GOTOs into loops and out of loops should be avoided, if possible.

Related instructions:

FOR-NEXT, REPEAT-UNTIL

Example:

```
100 ON ERROR GOSUB Error handling
200 REM Main program
990 END
1000Error handling:
1010 WHILE ERM(0)=73
1020    PRINT "Please check printer"
1030    REPEAT
1040       INPUT "Continue with 'j cr'",A$
1050    UNTIL A$="j" OR A$="J"
1060 WEND
1100 REM Handling of further errors
2000 RETURN
```

Graphics instruction

Drawing of Line Patterns

Purpose: This instruction allows the drawing of dashed lines or rectangles filled with a pattern.

The parameter is a 16-bit number in which each bit draws a dot. This pattern applies to the following DRAW, RDRAW and AREA instructions until it is replaced by another instruction.

Syntax: `[n] WIDTH a`

n: line number, also including label

a = -1: continuous line
BIN$(-1) → 1111111111111111

a = 21845: every second dot is drawn
BIN$(21845) → 0101010101010101

.
.
.

a = 255: dashed (half/half)
BIN$(255) → 0000000011111111

a = -1 the default value on power-up

Related instructions: `DRAW, RDRAW, AREA, SET, COLOR`

Graphics instruction

Relative Screen Coordinates, Display Area of Screen

Purpose: Determines the coordinate system for the graphics output.

Syntax:

```
[n] WINDOW x1,x2,y1,y2
```

n: line number, also including label
x1...y2 are numeric expressions for:

x1 : start x coordinate
x2 : end x coordinate

y1 : start y coordinate
y2 : end y coordinate

Remarks: After starting BASIC with RUN, WINDOW 0, 639, 0, 399 is initialized.

WINDOW refers to the coordinates for the DOT, MOVE, DRAW, RMOVE, RDRAW and AREA instructions.

Related instructions: VIEWPORT, ZOOM

Example: Display of a sine curve with relative coordinates.

```
110 WINDOW -PI,PI,-1,1
120 MOVE -PI,0
130 FOR X=-PI TO PI STEP .0628
140 DRAW X,SIN(X)
150 NEXT
```

Graphics instruction

Enlargement and Selection of Display Area

Purpose: This instruction enlarges a section of the graphics memory by a factor a. The contents of the graphics memory is not changed but each dot is repeated a times in the horizontal and vertical directions. The instruction is used to enlarge sections of the screen or to select the visible section of the graphics memory.

Syntax: `[n] ZOOM a`

n: line number, also including label
a: factor between 0 and 15

Remarks: The starting point for magnification is specified by the position of the graphic cursor. Enlargement is then made in the positive x direction and the negative y direction. The graphic cursor is then located at the top left corner of the screen following the instruction.

The cursor must be set to the original position in order to reduce to the original coordinates with ZOOM 0.

Adding graphics during enlargement is not allowed. The ZOOM command serves for a blow-up view of constructed graphics displays.

ZOOM only changes the visible section of the graphics memory; the coordinate system determined by VIEWPORT and WINDOW is not changed. Thus, it is possible to draw outside the visible section. By setting the cursor to the top left corner of this area (outside the visible area) and subsequently using ZOOM 0, this area is made visible.

The graphics memory includes a total of 1638 lines (dots in the vertical direction), but only 480 are just visible on the screen.

Note: The color graphics option (PCA-B3) permits only ZOOM 0 to be used.

Related instructions: `WINDOW, VIEWPORT`

3 BASIC Error Messages

Error message		Type of fault
ERROR 1:	*"hardware not installed"*	The interface addressed by the software is not fitted.
ERROR 2:	*"COM:DSR not active"*	The external V24 device is not ready or not connected to the PCA.
ERROR 3:	*"COM: timeout"*	Timeout for handshake or data transfer on V24 interface.
ERROR 4:	*"COM: overrun"*	The input buffer of the V24 interface contains more than 512 characters.
ERROR 5:	*"COM: parity"*	Error in test bit in data read by V24 interface.
ERROR 6:	*"COM: framing"*	Stop bit has not been received with data input to V24 interface.
ERROR 7:	*"device not open"*	An attempt has been made to read or write from an interface which has not been opened with 'open'.
ERROR 8:	*"device driver not installed"*	The device driver of the operating system accessed by the incorrectly terminated instruction has not been loaded in CONFIG.SYS.
ERROR 9:	*"subr. not loaded"*	A subroutine number has been output with CALL# which has not yet been loaded.
ERROR 10:	*"IEC-bus timeout"*	IEC-bus waiting time exceeded.
ERROR 11:	*"IEC-bus handshake error"*	Error in IEC-bus handshake with NDAC and NRFD.
ERROR 12:	*"not an IEC-bus talker/listener"*	BASIC is not in the IEC-bus talker/listener status.
ERROR 13:	*"not an IEC-bus controller"*	BASIC is not in the IEC-bus controller status and must therefore not execute the instruction.
ERROR 14:	*"I/O-control param. out of range"*	The string contains parameters outside the permissible range.
ERROR 15:	*"I/O-control syntax"*	String for setting the input/output interface is faulty.
ERROR 16:	*"device: general failure"*	Interface for input/output indicates an error.

ERROR 17:	*"device: time out"*	Input/output via the interface is not possible within the specified period of time.
ERROR 18:	*"too deeply nested"*	The maximum permissible number of subroutine levels has been exceeded.
ERROR 19:	*"FOR without TO"*	FOR NEXT loop is not completely defined.
ERROR 20:	*"redimensioned array"*	This data field has already been defined.
ERROR 21:	*"duplicate label"*	The marker has already been given away.
ERROR 22:	*"incompatible version"*	E.g. a device driver does not have required version number.
ERROR 23:	*"GOSUB without RETURN"*	Subroutine call without return instruction.
ERROR 24:	*"line number >65534"*	The highest line number 65534 has been exceeded.
ERROR 25:	*"undefined line or label"*	Branch instruction to a non-existent program line.
ERROR 26:	*"FOR without matching NEXT"*	The NEXT instruction is missing in matching the FOR NEXT loop.
ERROR 27:	*") and (out of balance"*	Incorrect combination of ()
ERROR 28:	*"undefined command"*	The entry does not correspond to an existing instruction.
ERROR 29:	*"out of memory"*	Permissible memory area violated.
ERROR 30:	*"undefined variable"*	Field variable used has not been previously dimensioned.
ERROR 31:	*"numeric overflow"*	Permissible numerical range of controller has been exceeded. e.g. when converting into an integer.
ERROR 32:	*"subscript out of range"*	Field index outside the permissible range limits.
ERROR 33:	*"illegal math. operation"*	Illegal mathematical operation
ERROR 34:	*"RETURN without GOSUB"*	Return from a subroutine not called by GOSUB.
ERROR 35:	*"syntax error"*	Faulty instruction; brackets, letters or characters are wrong.
ERROR 36:	*"lines nested"*	Illegal RENUMBER or CHAIN statement.
ERROR 37:	*"variable type mismatch"*	Wrong type of variable used.
ERROR 38:	*"undefined operator"*	Undefined variable within an instruction.

ERROR 39:	*"out of data"*	An attempt has been made to read in more data than are present.
ERROR 40:	*"parameter too large"*	Parameter outside the permissible range.
ERROR 41:	*"line too long"*	The permissible number of 80 characters within a line has been exceeded.
ERROR 42:	*"illegal quantity"*	The permissible range of the operation has been exceeded.
ERROR 43:	*"can't continue"*	Program cannot be continued with "CONT".
ERROR 44:	*"parameter out of range"*	The permissible parameter range has been violated.
ERROR 45:	*"CHAIN line erase"*	The CHAIN line has been overwritten by the program to be loaded.
ERROR 46:	*"Pascal not loaded"*	PASCAL cannot be loaded.
ERROR 47:	*"undefined Function"*	Function call (FN) without previous DEF FN.
ERROR 48:	*"no IF-struc., [ELSE] ENDIF match"*	The structure is not complete
ERROR 49:	*file already open"*	An attempt has been made to open a file which is already open.
ERROR 50:	*"file not open"*	An attempt has been made to use an unopened file.
ERROR 51:	*"DOS close error"*	Error when closing a file.
ERROR 52:	*"DOS open error"*	Error when opening a file, e.g. because it does not exist on this subdirectory.
ERROR 53:	*"DOS write error"*	Error when recording on floppy disk.
ERROR 54:	*"no valid file number"*	The reference number of the file is not within the permissible range.
ERROR 55:	*"file not open of output"*	The file has not been opened as an output file.
ERROR 56:	*"file not open for input"*	The file has not been opened as an input file.
ERROR 57:	*"file name error"*	The permissible length of the file name with 8 letters before the point and 3 letters after it has been exceeded.

ERROR 58:	*"DAC output overflow"*	Refers to the ANG interface
ERROR 59:	*"ADC input overflow"*	Refers to the ANG interface
ERROR 60:	*"WHILE without WEND"*	Structure not complete
ERROR 61:	*"WEND without WHILE"*	Structure not complete
ERROR 62:	*"REPEAT without UNTIL"*	Structure not complete
ERROR 63:	*"UNTIL without REPEAT"*	Structure not complete
ERROR 64:	*"DOS write protected disk"*	Floppy disk is write protected
ERROR 65:	*"DOS: unknown unit"*	*)
ERROR 66:	*"DOS: drive not ready"*	No floppy disk inserted in drive
ERROR 67:	*"DOS: unknown command"*	*)
ERROR 68:	*"DOS: data error"*	*)
ERROR 69:	*"DOS: bad requests s. length"*	*)
ERROR 70:	*"DOS: seek error"*	*)
ERROR 71:	*"DOS: unknown media type"*	wrong floppy disk format
ERROR 72:	*"DOS: sector not found"*	Defective floppy disk or Winchester
ERROR 73:	*"DOS: printer out of paper"*	Printer out of paper, not switched on or not selected.
ERROR 74:	*"DOS: write fault"*	*)
ERROR 75:	*"DOS: read fault"*	*)
ERROR 76:	*"DOS: general failure"*	*)
ERROR 77:	*"protected file loaded"*	the loaded program is not to be listed and cannot be changed
ERROR 78:	*"too many GOTO/GOSUBs"*	too much branching (memory overflow)
ERROR 79:	*"too many open files"*	Opening of further files or interfaces not possible
ERROR 80:	*"function not allowed here"*	variable name must not begin with FN (function call!)

*) Error messages of the operating system.

4 Applications

4.1 Program Transfer PUC → PCA

BASIC programs written on the PUC may be used on the PCA provided certain program modifications are carried out. This is because the BASIC syntax selected for the PCA has a certain uniformity in the make up of the instructions.

Differentiation is made between the following groups of BASIC instructions:

- **Identical instructions**

These BASIC instructions, e.g. the main part of the internationally accepted ANSI BASIC, can be run without modification on the PCA.

- **Similar instructions**

This group of instructions either has the same function as in the PUC, but with a different syntax, or the function of a command with unchanged syntax has been slightly modified as a result of development and new standards.

- **Fundamentally different instructions**

Parts of the program which contain instructions dependent on the hardware or the operating system must be rewritten for the PCA.

- **New instructions on the PCA**

They do not affect the program transfer but may increase the execution speed.

4.1.1 Similar Instructions

Those instructions of the PUC and PCA which are similar can easily be replaced by the new syntax on the PUC using the replace instruction (REP).

The following instructions can be converted:

	PUC		PCA
Printer V24	OPEN1,230 OPEN1,232		OPEN O #1,"LPT1:" OPEN O #1,"COM1:" OPEN I #1,"COM1:"
Floppy Disk Drive	OPEN1,1,0 "Name" OPEN1,1,1 "Name"		OPEN I #1,"Name" OPEN O #1,"Name"
Graphics		Kurzform	
	G.DOT, X,Y	(G.D)	DOT X,Y
	G.MOVE,X,Y	(G.M)	MOVE X,Y
	G.LINE,X,Y	(G.L)	DRAW X,Y
	G.WIDTH,a	(WID,a)	WIDTH a
	G.INVERT	(G.I)	INVERT
	G.WINDOW	(G.WIN)	WINDOW
	G.VIEWPORT	(G.V)	VIEWPORT
	G.PAGE	(G.P)	CLEAR
	G.SET,OFF	(G.S,O)	SET 0
	G.SET,INVERS	(G.S,I)	SET -1
	G.SET, ON	(G.S, ON)	SET -
	G.GCOPY	(G.G)	COPYOUT
IEC Bus	IEC DEV x		no longer required
	IEC OFF SRQ		ON SRQ RETURN
	IEC SRQ GOTO m		ON SRQ GOTO m
	IEC RET SRQ		RETURN
	IEC TIME x		IEC TIME x * 65
I/O	GET		INKEY
PRINT	?"S"	(Home)	"E_c[0;0H"
	?" *) "	(Clear Home)	"E_c[2J"
	?"Q"	(Cursor ↓)	"E_c[1B"
	?"P"	(Cursor ↑)	"E_c[1A"
	?"J"	(Cursor →)	"E_c[1C"
	?"I"	(Cursor ←)	"E_c[1D"
	?"R"	(Revers)	"E_c[7m"
	?"_"	(Off Revers)	"E_c[0m"

*) Heart graphics

4.1.2 Instructions to be Rewritten

This group includes all the instructions for accessing the floppy disk, since the floppy disk is now managed by the operating system. The data are filed and selected again using a program name instead of via the SET instruction. Direct files accessing the sector and byte via hardware are no longer possible with the PCA. The labelling of graphics must also be rewritten, which are frequently implemented in the PUC with POKE in the video memories or with cursor movements and PRINT instructions. The PCA possesses the far more convenient LABEL instruction which enables point-by-point addressing as well as 16 character sizes and 8 directions of tracing.

Examples of instructions which must be rewritten:

OPEN#1,	200...203,a
SET	
G.TAKE	(G.T.)
G.COPY	(G.C.)
PRINT	(Cursor movement)

This group also contains machine instructions, machine programs and BASIC instructions such as PEEK and POKE which use the hardware. Machine programs and programs containing machine routines must therefore be rewritten on the PCA using the assembler of the operating system and matched to the modified hardware addresses and components.

Machine subroutines are mainly required in the PUC for time-critical parts of the programs, e.g. adjustment procedures. Since the PCA has a higher computing speed as well as new BASIC instructions, it should be examined if the programming problem may be solved using BASIC rather than resorting to a subroutine in machine language.

Examples of machine instructions which cannot be executed on the PCA:

POKE
PEEK
SYS
USR

4.1.3 Transfer of the Program from PUC to PCA

A program which is run on the PUC is transferred to the PCA using the following steps:

- First, all similar instructions are changed on the PUC into the PCA syntax using REP.
- All parts of the program which contain instructions, that cannot be converted are then written on the PUC. The program must be sufficiently free from errors so that the PCA does not detect any syntax errors during loading. Lines containing syntax errors are not transferred to the BASIC program on the PCA.
- The actual transfer takes place from a PUC, model 10 via the IEC bus. To this end, both devices must be connected via an IEC-bus cable. MS-DOS and BASIC as well as the program YFER.BAS contained in the subdirectory \USER on the system floppy disk are first loaded into the PCA.

  Following the start, the PCA waits for a name to be entered under which the transferred program is to be filed on the floppy disk as an ASCII file. Following input of the name, the PCA waits to be addressed as a listener.

  The following instruction is then entered on the PUC, model 10:

```
IECTIME100: IECLIST1.
```

  If everything is carried out correctly, the program will then be listed on the PCA screen.

- The program is simultaneously filed on the floppy disk as an ASCII file. It can be loaded into the PCA as an executable BASIC program after entering NEW with

  ```
  ALOAD"NAME.ASC"
  ```

  NAME is the program name specified above. If errors were present in the program, the error beep sounds with each error and the error message appears in the status line. It is recommendable to reedit on the PUC and to repeat the procedure if many errors occurred.

- The PCA signals READY if no errors have been detected during ALOAD. The program now contains no more syntax errors. This, however, does not mean that it is error-free. It is now possible to further test and edit the program on the PCA.

4.1.4 Further Instructions

The following differences are, in addition, to be found in the PCA:

- PRINT, (PRINT comma) is not permissible
- Fields with less than 10 parameters must be dimensioned
- A field variable must not be used as index for fields.

4.2 Matrix Module MATRIX.BAS

The BASIC program module MATRIX.BAS is a complex software routine contained in the USER on the PCA system floppy and is used to

- determine the result vector of an n-dimensional system of equations
- calculate the inverse values of a quadratic n-dimensional matrix
- multiply two (n*m)-dimensional and (m*1)-dimensional matrices
- divide two (m*n)/(m*m) quadratic n-dimensional matrices

The individual operations can be selected using the module sector X (= 1,2,3,4) where the entry point is always 51000.

Example: Matrix inversion

```
X=2:GOSUB 51000
```

The basic software "MATRIX.BAS" is supplemented on the system floppy by a demo program which clarifies the input parameters required for each individual operation.

It should be noted that the R&S BASIC predefines the dimensioning of each vector and field used.

In the case of multi-dimensioning, all variables must therefore be first deleted, if necessary, using the instruction CLR (see e.g. line 76).

4.2.1 Gaussian Algorithm: X = 1

This algorithm solves an n-dimensional system of equations in the form

[A] * X] = B] Evaluation (1)

for the unknown n-dimensional vector X].

Input parameters:

N1	Dimension n of the system of equations
A1(N1,N1)	Matrix [A] according to equation (1)
A1(N1,N1 + 1)	Vector B] according to equation (1)
IT	Numerically dependent iteration limit depending on the digit accuracy of the computer used IT ≈ 5E-14 for the PCA5

Output parameters:

X1(N1)	Required vector X] according to equation (1)
ER$	"linear dependancy" Error message with linearly dependent system of equations
ER = 1	Error flag for further evaluation

Module-internal variables:

K1, U1, I1, J1

Example:

$$\begin{bmatrix} 2 & -1 & 2 \\ 3 & 4 & -2 \\ 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} -9 \\ 28 \\ -7 \end{bmatrix}$$

Solution:

$$X] = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ -5 \end{bmatrix}$$

4.2.2 Matrix Inversion: X = 2

With X = 2, any quadratic n-dimensional matrix [A] can be inverted according to the relationship in equation (2)

[B] = [A]
[A] * [B] = [B] * [A] = [E] Evaluation (2)

The input matrix [A] is not changed in the process.

The algorithm used requires an auxiliary matrix [A1] and the auxiliary vector X1] which must also be dimensioned.

Input parameters:

N1	Dimension n of matrix [A] according to equation (2)
A2(N1,N1)	Matrix [A] according to equation (2)
A1(N1 + 1,N1 + 1)	Auxiliary matrix [A1]
X1(N1 + 1)	Auxiliary vector X1]
IT	Iteration limit (see Gaussian algorithm)

Output parameters:

A3(N1,N1)	Inverted matrix [B] according to equation (2)
ER$	Singular matrix Error message if determinant [B] = 0
ER = 2	Error flag for further evaluation

Module-internal variables:

K2, I1, J1, K1, U1

Example:

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}^{-1} = \begin{bmatrix} -2.5 & 1.5 \\ 2 & -5 \end{bmatrix}$$

4.2.3 Matrix Multiplication: X = 3

Using this routine, any two matrices with the dimensions (n*m) and (m*e) are multiplied together.

The input matrices [A] and [B] are not changed in the process.
The calculation is as follows:

[C] = [A] * [B] Equation (3)

If [A] is an (n*m) matrix and [B] an (m*e) matrix, the resulting matrix [C] has the dimension (n*e).

The number of columns and lines in the matrices [A] and [B] must be identical (see error flag ER).

Input parameters:

Z1	Number of lines in matrix [A] according to equation (3)
S1	Number of columns in matrix [A] according to equation (3)
Z2	Number of lines in matrix [B] according to equation (3)
S2	Number of columns in matrix [B] according to equation (3) caution S1 = Z2!
A4(Z1,S1)	Matrix [A] according to equation (3)
A3(Z2,S2)	Matrix [B] according to equation (3)
IT	Iteration limit (see Gaussian algorithm)

Output parameters:

A6(Z1,S2)	Resulting matrix [C] according to equation (3)
ER$	"non fitting matrix dimensions" Error message if S1 <> Z2
ER = 3	Error flag for further evaluation

Module-internal variables:

I1, K1, J1

Example:

$$\begin{bmatrix} 1 & -2 \\ 5 & 0 \\ 3 & -5 \end{bmatrix} * \begin{bmatrix} -2 & -5 & -3 & 1 & 2 \\ 2 & 1 & 2 & 1 & -2 \end{bmatrix} = ?$$

Solution:

$$\begin{bmatrix} -6 & -7 & -7 & -1 & 6 \\ -10 & -25 & -15 & 5 & 10 \\ -16 & -20 & -19 & -2 & 16 \end{bmatrix}$$

4.2.4 Matrix Division: X = 4

The matrix division is based on the multiplication of an inverted matrix according to equation (4).

[C] = [A]/[B] = [A]*[B] Equation (4)

It is therefore only possible to divide a quadratic matrix [B] which has the dimension n of the number of columns in matrix [A].

The algorithm internally requires the auxiliary matrix [A1] and the auxiliary vector X1] which must also be dimensioned.

Input parameters:

Z1	Number of lines of matrix [A]
S1	Number of columns of matrix [A]
N1	Dimension of matrix [B] according to equation (4)
	Caution: N1 = S1 (see error flag)
A4(Z1,S1)	Matrix [A] according to equation (4)
A2(N1,N1)	Matrix [B] according to equation (4)
A1(N1 + 1,N1 + 1)	Auxiliary matrix [A1]
X1(N1 + 1)	Auxiliary vector X1]
IT	Iteration limit (see Gaussian algorithm)

Output parameters:

A6(1,N1)	Resulting matrix [C] according to equation (4)
ER$	"singular matrix"
	Error message if determinant [B] = 0
ER = 2	Error flag for further evaluation
ER$	"non fitting matrix dimensions"
	Error message if S1 < > N1
ER = 4	Error flag for further evaluation.

Module-internal variable:

K2, I1, K1, J1, U1

Example:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \div \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 1.5 & -0.5 \\ 0.5 & 0.5 \\ -0.5 & 1.5 \end{bmatrix}$$

4.3 Graphic Examples "GPH-PCA.BAS" or "GPH.ASC"

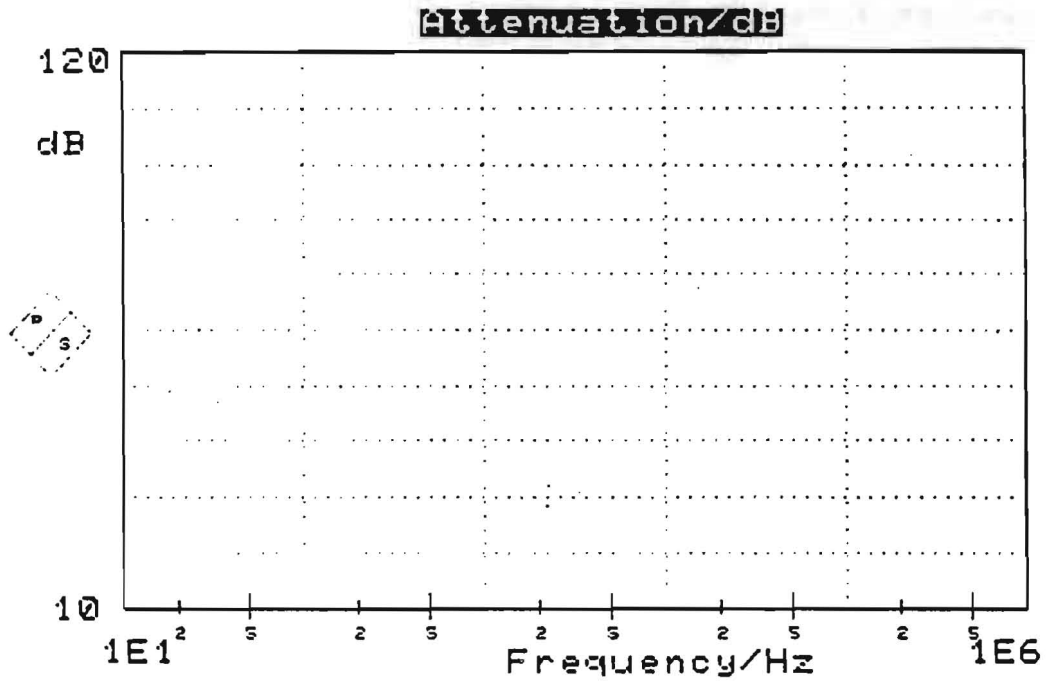
The universal BASIC graphic module "GPH-PCA.BAS" is suitable for many representations of measured results in linear or logarithmic scales. It is particularly convenient and easy due to the plotting routine for the measured values, which saves the user the need to study the PCA graphics command set and the address calculation that proves rather complicated in the case of logarithmic representations.

How is the subroutine "GPH-PCA.BAS" operated? The routine 'GOSUB 2500' or 'GOSUB Graf_init' (= building up a graphic representation) requires the following parameters to be entered:

XL : = min. value of x-axis;
XH : = max. value of x-axis;
YL : = min. value of y-axis;
YH : = max. value of y-axis;
SX : = number of divisions of x-axis;
SY : = number of divisions of y-axis;
LY$: = label y-axis (max. length: 8 characters)
L$: = heading
LX$: = label x-axis
LG : = switch lin/log
LG = 0 linear x-axis/linear y-axis
LG = 1 logarithmic x-axis/linear y-axis
LG = 2 linear x-axis/logarithmic y-axis
LG = 3 logarithmic x-axis/logarithmic y-axis

Example 1:

```
10 LG=1
20 XL=10:XH=1E6
30 YL=10:YH=120
40 LY$="dB"
50 L$="Attenuation/dB"
60 LX$="Frequency/Hz"
65 GOSUB Graf_init
66 END
```



The above mode of representation may be extended as follows:

FM : = marker frequency
TL : = number of tolerance curves

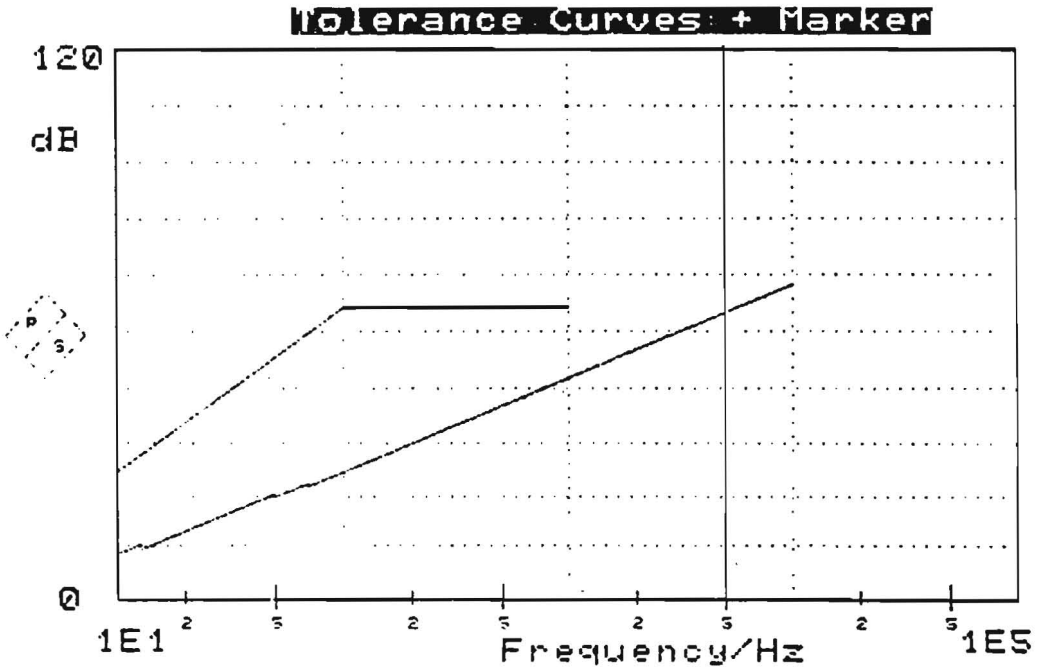
If TL>0 the number of reference values TS (TL) per tolerance curve and the X/Y values per reference value must be indicated for each of these tolerance curves.

F (TL, TS (TL)) : x-value
W (TL, TS (TL)) : y-value

The variables TS (TL), G (TL, TS (TL)) and W (TL, TS (TL)) have to be dimensioned before the program is started.

Example 2:

```
10 DIM TS(2),F(2,4),W(2,4)
20 XL=10:XH=1E5:FM=5000
30 XL=0:YH:120
40 LY$="dB"
50 L$="Tolerance Curves+Marker"
60 LX$="Frequency/Hz"
65 TL=2:TS(1)=4:TS(2)=3←
70 F(1,1)=10:W(1,1)=10:F(1,2)=100:W(1,2)=30
75 F(1,3)=1000:W(1,3)=50:F(1,3)=10000:W(1,4)=70
80 F(2,1)=10:W(2,1)=30:F(2,2)=100:W(2,2)=65:F(2,3)=1000:W(2,3)=65
90 GOSUB Graf_init
95 END
```

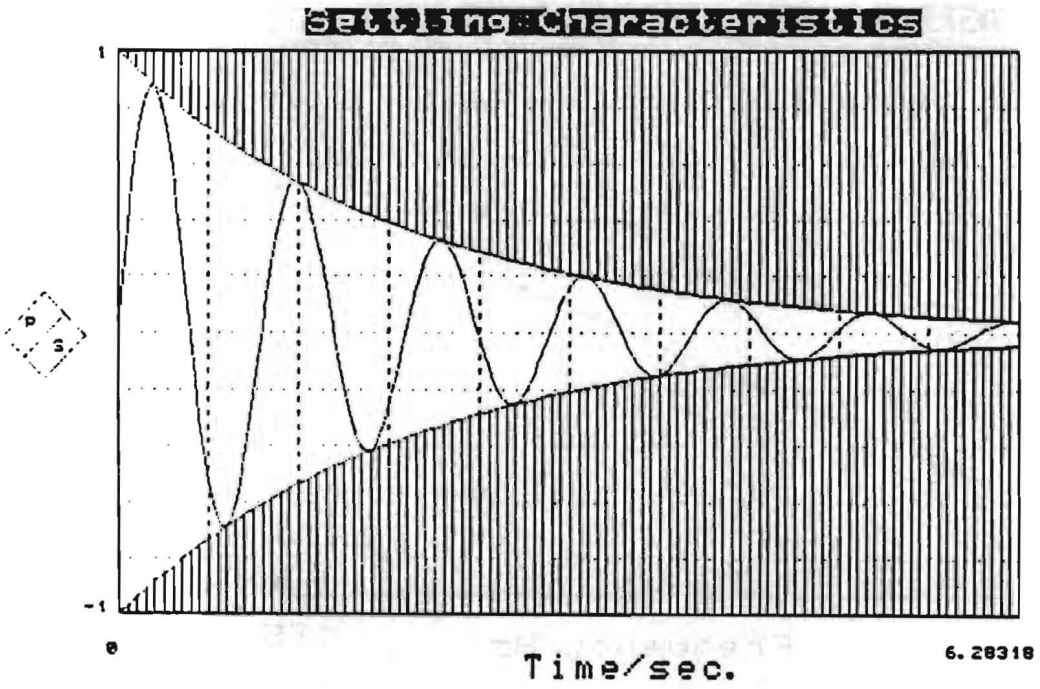


The subroutine 'GOSUB 25085' or GOSUB Graf_exe' graphically presents the pair of measured values (XW, YW) on the screen. It may be presented linearly (LG = 0), logarithmically (LG = 1; x log/y lin; LG = 2: x lin/ y<log) and double logarithmically.

If more than one measured curve is to be included in a graphic representation the variable GD will have to be set to '0' (GD = 0) before each output of a new curve. Thus, the graphic cursor is brought back to the origin.

Example 3:

```
100 PI=4*ATN(1)
110 LG=0:TL=0
120 LY$="Volt":L$="Settling characteristic":LX$="Time/sec."
130 XL=0:XH=2*PI
140 YL=-1:YH=1
150 GOSUB Graf_init
160 SF=3:GD=0←
170 FOR XW=XL TO XH STEP XH/100
180 YW=EXP(-XW/2):GOSUB Graf_exe
185 NEXT
190 SF=2:GD=0←
200 FOW XW=XL TO XH STEP XH/100
210 YW=-EXP(-XW/2):GOSUB Graf_exe
220 NEXT
230 SF=0:GD=0←
240 FOR XW=XL TO XH STEP XH/200
250 YW=SIN(2*PI*XW)*EXP(-XW/2):GOSUB Graf_exe
260 NEXT
270 END
```

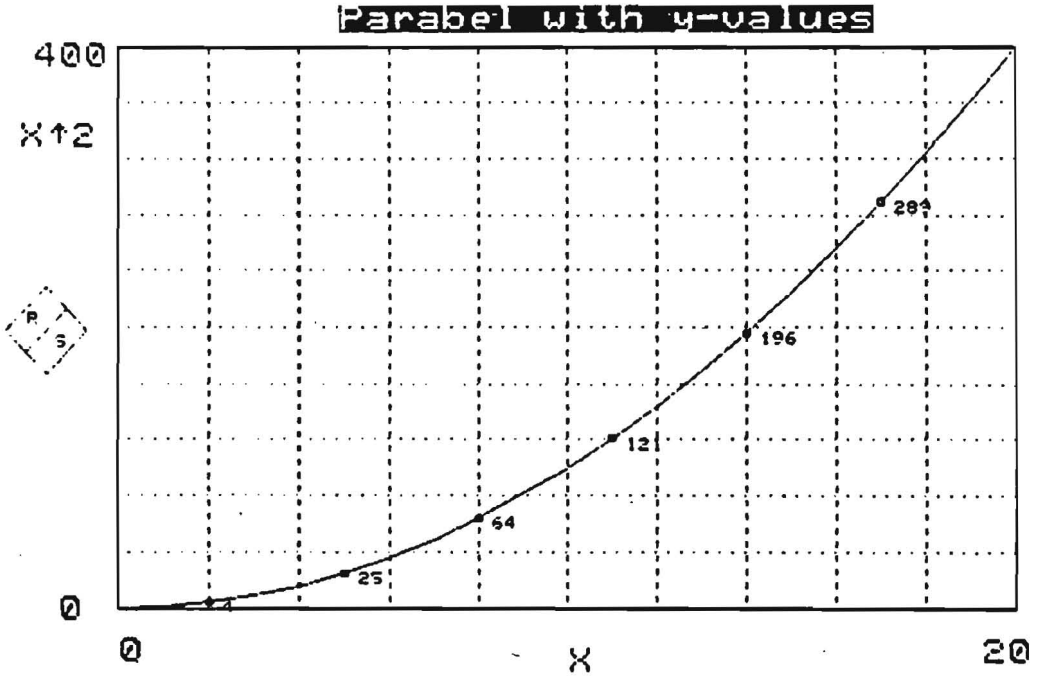


For some applications, it is desirable to also indicate the definite numeric value of the measured value in the graphics.

By means of determining the GPH-PCA variables Py = m, each m-value of y is shown in the graphics.

Example 4:

```
100 LG=0: SF=0: PY=3
110 XL=0: XH=20:YL=0: YH=400
120 LY$="X^2": LX$="X": L$="Parabel with Y-values": GOSUB Graf_init
130 FOR XW=XL TO XH: YW=XW^2: GOSUB Graf_exe NEXT
140 END
```

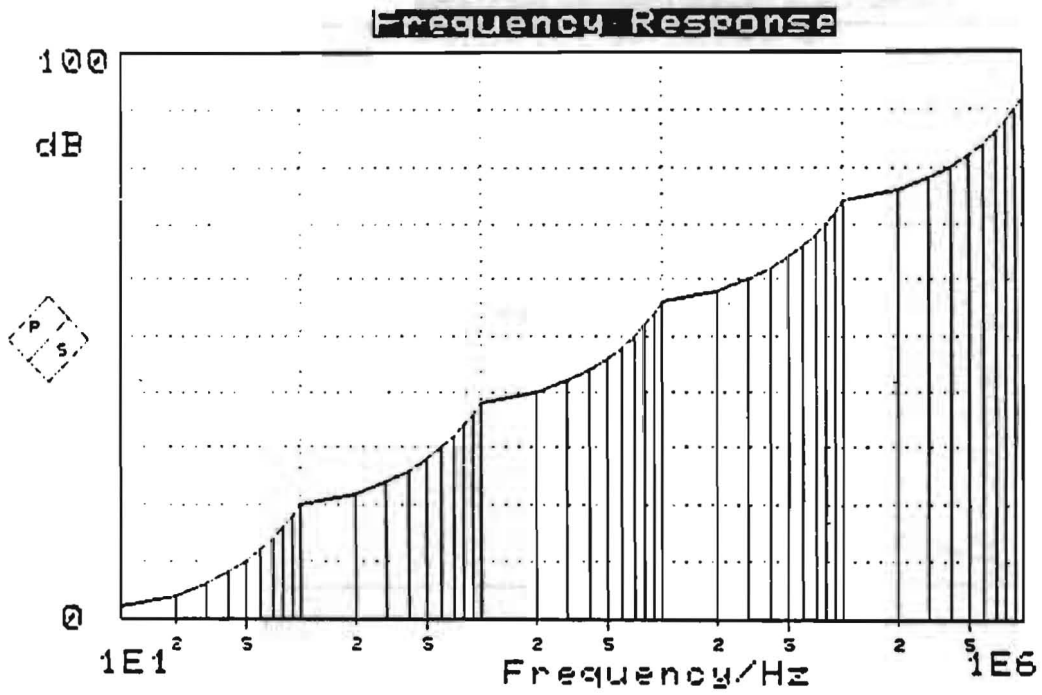


If problems in measuring arise (mostly in the frequency domain) which necessitate a logarithmic representation, a variable step width will often be required.

By determining the GPH-PCA variables SD = n, the step width SW of the x-axis is automatically calculated in such a manner that n measurements per decade result.

Example 5:

```
10 LG=1:SD=10←
20 XL=10:XH=1E6
30 YL=0:YH=100
40 LY$="dB"
50 L$="Frequency Response"
60 LX$="Frequenz/Hz"
70 GOSUB Graf_init
80 XW=XL:YW=0GD=0:SF=2
85 XW=YW+2:GOSUB Graf_exe
88 XW=XW+SW:IF(XW-XH)<0THEN85
90 END
```



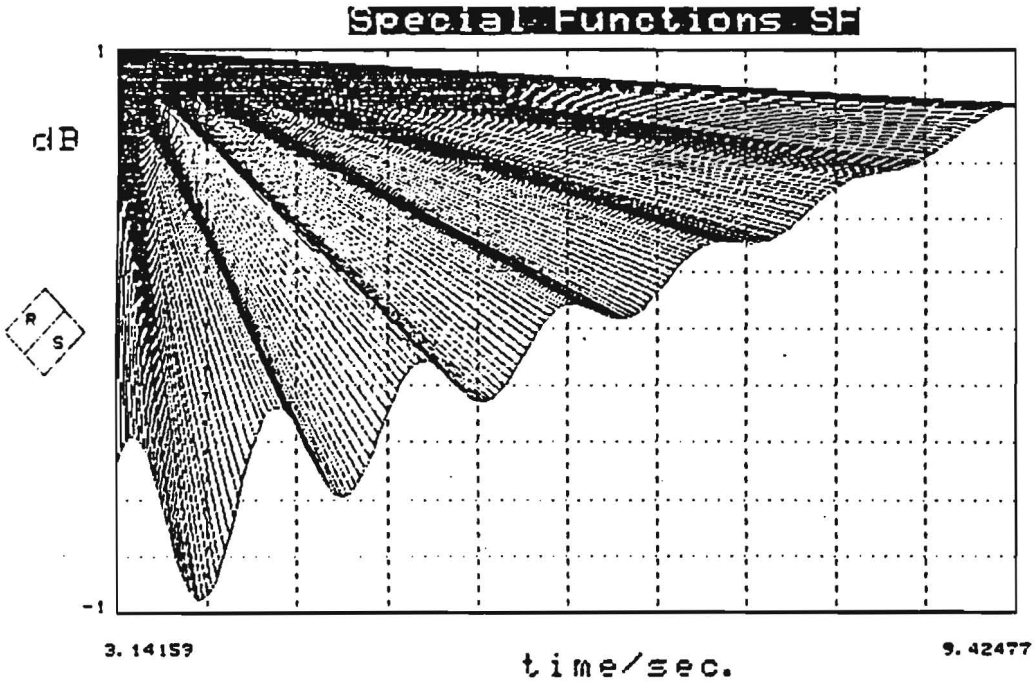
The representation of the measured curve may finally also be modified by special functions SF = 0 to 9. SF = 0 is the normal form of representation.

If SF > 0, the measured values are vertically extended by lines in direction of the x-axis. The effect of SF is shown in the following figure.

See also examples 3 and 5.

Example 6:

```
1    PI=4*ATN(1)
10   LG=0:SF=7←
20   XL=PI:XH=3*PI
30   YL=-1:YH=+1
40   LY$="dB"
50   L$="Special Functions SF"
60   LX$="Time/msec."
70   GOSUB Graf_init
75   FOR XW=XL to XH STEP XH/300
76   GOSUB 1000:GOSUB Graf_exe
78   NEXT
85   END
1000 YW=2*SIN(2*PI*XW*EXP(-XW/2))-.9+XW
1010 XW=XW+.008
1020 RETURN
```



Color Representations on GPH-PCA

Representations in color are possible by setting the GPH-PCA variables CO = 1 and the corresponding hardware (PCA2/12 including option -B3 and PMC).

If CO = 1, the following color combinations are used:

color of coordinate system:	white
color of label:	blue
color of tolerance curves:	green
color of marker frequency:	red
color of measured curve:	yellow

If other color combinations are needed, a broad range of colors may be obtained by setting the following variables:

CA : = color pen for coordinate system
CL : = color pen for label
CT : = color pen for tolerance curves
CM : = color pen for marker frequency
CC : = color pen for measured curve

The definition of the color pens is effected by means of the COLOR-command. Of course, also the default color table may be used.

If for instance several measured curves are to be included into a diagram in different colors, it will be necessary to modify the variable CC before each measured curve (GD = 0!) is drawn.

Output of Graphics on Plotter or Printer

The GRAPHIC instruction of the BASIC Interpreter is a convenient means for activating the plotter as the medium for output.

Example:

```
10 GRAPHIC "GRAPH", "DOP"
```

The following graphic instructions are executed both on the screen and on the Plotter DOP.

The graphic module offers a convenient routine for outputting the produced graphics on PUD2/3. The following call is required:

```
GOSUB 25146 or GOSUB Hardcopy
```

By means of setting the GPH-PCA variables SP either an immediate output may be selected or an additional inquiry (printer output yes/no) may be faded in via a menu.

The variable SP is to be defined as follows:

SP = 1 hardcopy without inquiry
SP<>1 hardcopy with previous inquiry

Note:

The above output routines are used by the command "ON ERROR" for detecting a wrongly connected printer. Other user specific "ON ERROR" routines need to be reactivated following the selection of this routine.

If a hardcopy with previous inquiry is selected the PCA softkeys will be assigned their meanings by the graphic module. Following the exit from the routine the standard softkey assignments of the BASIC interpreter are set.

4.4 BASIC Compatibility of PSA/PAT Controllers in Comparison with PCA Controllers

Most of the tested BASIC programs run on the PSA/PAT without modifications. There are no compatibility problems as far as the IEC bus is concerned and as far as graphics is concerned only the ZOOM instruction entails a few problems, such as output of the display on the screen. These problems are caused by the different graphics hardware and can be avoided by considering the following items:

1) On the PSA/PAT screen which is compatible to industrial standard either (high resolution) graphics or alphanumeric characters can be displayed, but not simultaneously and not superimposed which is possible on the PCA. Deleting the graphics causes deletion of the text and vice versa.

2) 30 lines can be output on the screen only in VGA mode (SCREEN 17 or 18). In all other modes (CGA, EGA and HERCULES) only 25 lines can be displayed.

3) The graphics memory can store only 1 page (VGA, CGA) or 2 pages (EGA, HERCULES mode) in contrast to the PCA, which stores up to 3 $^1/_3$ pages. These pages cannot be continuously scrolled or zoomed.

For compensating these differences note the following:

Re 1:

After loading BASIC and upon RUN the device is in graphics mode; depending on the graphics adapter and the monitor connected the VGA, EGA HERCULUS or CGA mode is selected black/white or colored. The new SCREEN instruction allows for switching to another mode.
Caution: for the HERCULES mode the support routines must be resident. They are loaded using the MS-DOS command HERCSUP.COM. A few graphics multifunction boards must first be initialized for this mode by the HERCMOD command).

The Escape sequence "Ec[j" or the SCREEN 3 instruction may be used for switching to the alphanumeric mode, if the program consists exclusively of alphanumeric outputs. Of course alphanumeric characters can be output in graphics mode as well using the PRINT instruction but this mode is a bit slowlier than the alphanumeric mode and attributes are not allowed.

Each graphics instruction in the program automatically switches over to the graphics mode.

The F8 key allows for alternate switching between the alphanumeric and the graphics mode.

Restrictions: If the bottom of the screen is reached by the alphanumeric output, the graphics is scrolled. Graphics labellings with PRINT (instead of LABEL) do not correspond to the drawing, since the graphics coordinate system holds only 21 lines instead of 25 (except for VGA mode).

In all modes the graphics are displayed in the same size, since the VIEWPORT instruction always assumes a virtual monitor of 640 x 480 pixels like PCA but the resolution is lower than that of the PCA except for VGA mode 17, 18.

Re 2:

For the alphanumeric display of 25 lines (30 lines are only available in VGA mode which is compatible to the PCA 30-line mode!) the bottom line (line 25) is used as status and error line, allowing the user to write to 24 lines. If the status or softkey lines are labelled by the Escape sequences, 4 lines are reserved (instead of 5 in case of PCA);thus, 21 lines are still available (instead of 25 on the PCA).

Absolute positioning of the cursor below the defined range sets the cursor to the bottom line (line 20 to 23), a following LF scrolls the screen.

Re 3:

In the EGA and HERCULES modes two pages can be stored. Only in these modes background drawing is allowed. As soon as the y-coordinate is outside the visible range the second screen page is written to. MOVE and ZOOM select the visible page. A switch-over is initiated by y-coordinates above or below the visible screen. Continuous scrolling is not possible.
The INVERT instruction (invert b/w monitor) is not implemented at present.

The status and softkey lines are labelled according to the following table:

Screen lines	PSA/PAT-controller CGA, EGA, HERC.mode	PCA and PSA only VGA graphics mode
first	0	
↓	1	
	:	
	:	
		25 Q1
	21 Q1, Q3	26 Q2, Q
↓	22 Q2, Q4, Softk.	27 Q3
last	23 Q5	28 Q4, softk.
	24 Q	29 Q5

The ESC sequences

[a switch-over to 25 lines
[b switch-over to 30 lines

are only relevant for the VGA mode. In the other modes switching over to 21 lines is effected by pressing a and to 25 lines by pressing b (the space reserved for softkeys and status information is available again).

The following attributes are only available in alphanumeric mode:

Sequenz	Function		restricitons remark
[81m	display page 1		not in HERC.mode
[82m	display page 2		not in HERC.mode
[83m	write page 1		not in HERC.mode
[84m	write page 2		not in HERC.mode
[0m	reset attributes -		
[1m	increased intensity		
[2m	reduced intensity		normal intensity
[4m	underline		not in HERC.mode
[5m	slow blinking		normal blinking
[6m	fast blinking		normal blinking
[7m	reverse display		
[9m	switch on block graphics		—
[91m	lowest intensity	blue	intensity degrees correspond to colors (also in graphics mode!) but not in HERCULES mode
[92m		green	
[93m	↓	cobalt blue	
[94m		red	
[95m		violet	
[96m		yellow	
[97m	highest intensity	white	
[98m		white	
[:m	IBM character set		cancelled
[;m	PCA character set		cancelled
[>m	double character width		cancelled

The following sequences have been added:

[j switches over from graphics mode to alphanumeric mode
[1j leaves the PCA compatibility mode (keypad codes)
[2j switches over from alphanumeric mode to graphics mode

The following BASIC routine checks whether the PSA/PAT hardware is running and if running sets the IBMFLAG to 1.

```
100 SEGMENT 79
110 IF PEEK(0) + PEEK(1)=146 THEN IBMFLAG=1
(120 SEGMENT DEF)
```

ROHDE&SCHWARZ

BASIC-Interpreter Instruction Set

Version 2

PD 756.8210.24

Definitions of Terms Used

Character	Meaning
n, m	Line numbers (1 to 65.534)
t	String of characters
z	One-digit constant
k	Numeric constant
k%	Integer constant (-32.768 to 65.535)
vn	Numeric variable
v%	Integer variable
v$	String variable
v	One of the variables vn, v%, v$
a, b, c, x, y	Constant, variable or numeric expression
s$	String expression (string constant, variable or function)
a/s$	Numeric expression a or string expression s$
k/t	Numeric constant k or string constant t
(); =, # <> $: " %	Characters belonging to the syntax which must therefore be written
[ ]	Brackets which enclose a part of the statement. These are extensions to an instruction which are possible but not absolutely necessary.
[ ]...	Specified extensions may be repeated in the statement.

Instructions	Components of the program; are located immediately after a line number or a colon. If the instruction is written without a line number, it is executed immediately like command (direct mode).	PRINT GOTO
Graphics statements	Concern single dot graphics.	
IEC statements	Concern the IEC-625 bus.	
Commands	Cannot be components of a program since they generally handle programs. Their execution is always immediate.	ALOAD NEW
Functions	Always possess an argument.	
	Functions with a subsequent $ character always produce a character string as the result. (string function).	CHR$(10) MID$(A$)
	Functions without a subsequent $ character produce a numeric value as the result (numeric function).	SIN(A)
Synonyms	Notations of an instruction which are accepted as being compatible but are automatically converted internally by the controller into the PCA syntax.	IECATN IECATT
Default values	Parameters which the PCA uses if no parameters are specified in the statement.	

Uppercase letters	Mark keywords which must be entered in this sequence. The input may be both in upper-case and lower-case notation.
Lowercase letters	Dummy values for characters or character sequences freely selectable by the user. (The first letter of the examples is written in accordance with German notation).
Variable name	Any sequence of letters and digits as well as underline characters used as a means of differentiation. The first character must be a letter. Both upper-case and lower-case letters can be entered; BASIC always converts the first letter to upper case and the subsequent to lower case. A1, A2 Long_name
Labels	Jump targets of the GOTO/GOSUB instructions, the name establishing the reference. Labels are located after the line number and end with ":". The definition of the variable name given above also applies to labels. 100 Subroutin_no1:

Numerical functions and operators

>		greater than
<		less than
>	=	equal to or greater than
<	=	equal to or less than
<>		not equal to
=		equal to

(Relational operators)

AND	Boolean operations
OR	
NOT	
XOR	

SGN	(a)	Sign
INT	(a)	Integer
ABS	(a)	Absolute value
SQR	(a)	Square root
SIN	(a)	Angular functions
COS	(a)	
TAN	(a)	
ATN	(a)	
LOG	(a)	Logarithm to base e
EXP	(a)	Exponent to base e
a ↑ b		Power function
RND	(a)	Random function
ERM	(a)	Error poll
ERL	(a)	Poll of error line
FRE	(a)	Poll of freely available storage space
DEF FN		Functions definable by the user

Data

DATA k_1[, k_2]...	Setting up a data sequence
DIM v (a_1[, a_2]...)	Field dimensioning
INKEY v$	Keyboard poll
INPUT["t";]v_1[,vn]...	Keyboard input
INPUT$ (n,[#a])	String input with number of characters
READ v_1[,vn]...	Reading in data characters
RESTORE [m]	Reset data pointer

Program execution

BREAK OFF	Disable Break key
BREAK or EXIT	Enable Break key
BYE	Switch to the operating system
CLR	Set basic status of BASIC
CONT	Program continuation
END	Program end
ERASE v_0[, v_1]...	Clear variables
HOLD a	Waiting time in ms
REM	Remark
RUN [n]	Program start
SHELL [s$]	Call-up of MS-DOS commands
STOP	Program stop
TRACE	Output of line number of running program
TRACE OFF	Switch off TRACE mode
TRACE a/s$[,a/s$]...	Program execution in steps with output

Pseudo variables

DATE$	Read out date
DATUM$	Read out date (German)
PI	Circle constant
TIME	Measure or calculate time
TIME$	Read out time

Jumps and loops

IF THEN
ELSE
ENDIF
	Structure element (over several lines)

FOR vn = a TO b[STEP c] NEXT [vn]	Loops
GOSUB n	Jump into subroutine
ON a GOSUB n[, m]...	Jump depending on a
GOTO n	Unconditional jump
ON a GOTO n[, m]...	Jump depending on a
IF a THEN...ELSE	Comparison
ON ERROR GOTO n ON ERROR GOSUB n	Enable jump or subroutine call in case of error
ON COMa GOTO n ON COMa GOSUB n	Enable jump or subroutine call upon end-of-file character in interface
ON KEY GOTO n ON KEY GOSUB n	Enable jump or subroutine call upon keystroke
ON SRQ GOTO n ON SRQ GOSUB n	Enable jump or subroutine call upon Service Request
ON TIME GOTO n ON TIME GOSUB n	Enable jump or subroutine call at a given time
ON TTL v GOTO n ON TTL v GOSUB n	Enable jump upon TTL interrupt
RETURN [a]	Return from subroutine
REPEAT UNTIL a	Loop structure (over several lines) condition at the end
WHILE a WEND	Loop structure (over several lines) condition at the beginning

Character string processing

ASC(s$)	Conversion of ASCII character into numeric value
BIN(s$)	Conversion of binary numbers
BIN$(a)	and decimal numbers
CHR$(a)	Conversion of numeric value into ASCII character
HEX(s$)	Conversion of hexadecimal
HEX$(a)	and decimal numbers
LEFT$(s$, a)	Separate first characters from string
LEN(s$)	Length of a string
MID$(s$, a, b)	Remove middle characters from string
RIGHT$(s$, a)	Separate last characters from string
STR$(a)	Conversion of numeric variable into string
VAL(s$)	Conversion of string into numeric variable
+	Linking of character strings

Edit instructions

AUTO [n],[Δn]	Automatic line numbering
DELETE n-m	Delete lines
DIR [t]	Output directory
FRE(0) FRE(1)	Available memory space
HELP[arg]	Select and display support information texts
IEC[k]LIST ON a	Program output on IEC bus
OFF	Termination of program output on IEC bus
LIST [n][-[m]]	Program output on screen
NEW	Delete program
PLIST [n][-[m]]	Program output via printer interface
PRINT FRE (0)	Available data storage space
PRINT FRE (1)	Available program storage space
RENUMBER [m_1][-[m_2]][, n[,Δn]]	Renumbering of lines
SEARCH [n-m], t	Search text lines
SOFTKEY	Restore softkey labelling

Machine instructions

CALL a[, vn_1]...	Machine program call
CALL # a[, vn_1]...	Call machine routine
INP (a)	Read via I/O addresses
LOAD # a, s$	Load machine routines
OUT a, b	Output via I/O addresses
PASCAL a[, vn_1]...	Call Pascal routines
PEEK(a)	Read memory location
POKE a, b	Write into memory location
SEGMENT a/DEF	Fix a segment
VARPTR (v)	Read in a variable pointer

Graphics instructions

AREA x,y	Draw filled in rectangle
CLEAR	Clear graphics display
COLOR f, r, g, b	Color assignment
COPY OUT [a]	Output graphics to printer
DOT x,y	Draw dot
DRAW x,y	Draw line
GLOAD "s$"	Load graphics display from file
GRAPHIC s$[,s$]	Select graphics interface
GSAVE "s$"	Store graphics display on file
INVERT	Invert graphics display
LABEL s$ [, a[, b[, c]]]	Labelling of graphics
MOVE x,y	Position cursor
POLYLINE a,v%(b)	Draw polyline
RMOVE x,y	Position cursor relative
RDRAW x,y	Draw line relative
SET a [, b] [, c]	Display mode for lines and dots
VIEWPORT x1, x2, y1, y2	Fixing display area of screen
WIDTH a	Draw broken lines
WINDOW x1, x2, y1, y2	Fixing coordinate range
ZOOM a	Enlargement and selection of display area

Input/output via floppy disk, hard disk and interface

ALOAD"t"	Load program stored in ASCII code
ASAVE"t"	Save program in ASCII code
CHAIN s$, m	Reload program sections
CLOSE # [a][, a]...	Close file
FORM [m-n]	Set page format of printer
INPUT$(c[, # a])	Read in string via keyboard or interface
INPUT # a, v_1[, v_2]...	Load file
LOAD s$[, R]	Load program
OPENI # a, s$	Open input file
OPENO # a, s$	Open output file
OPENI # a,"CON:"	Input/output on console
OPENI #a, "COMb: B, P, A, S, T, E, C, H"	Input/output on V24/RS 232
OPENO # a,"LPT1:"	Output on printer
PLAY s$[, a]	Signal tone with pitch and duration in s$ and repetition rate a
PRINT # a, a/s$	Store data
PRINT list[;] or [,]	Character output on screen
PRINT USINGs$, list	Formatted output on screen
SAVE s$	Store program
TAB (a)	Distance from left edge of screen

A/D Converter

ADC a BLOCKIN "A, counter, channels, sample_rate [,Ttrigger]",v%(b)	Measure block with channel number and sampling rate; start measurement with digital trigger
ADC a BLOCKIN "A, counter, channel, sample_rate [Strigger]",v%(b)	Measure block with adjustable sampling rate; start measurement with analog trigger
ADC a IN "A,channel",v	Measure a channel value
ADC a IN "D",v	Read in the four digital input lines
ADC a OUT "D",v	Output to the four digital output lines

Analog I/O Interface

ANGa BLOCKIN "counter[, samplrate]", v%(b)	Measure block with selectable sample rate
ANGa BLOCKOUT "counter, repetitions, output_rate", v%(b)	Output block of values to D/A converter with selectable output rate
ANGa IN "U, m, b, g", v	Measure voltage with analog interface
ANGa IN "I, m, b, g", v	Measure current with analog interface
ANGa IN "R, b, g", v	Measure resistance with analog interface
ANGa IN "CAL x", A	Correct offset of voltage resistance measuring path of analog interface option
ANGa OUT "D", A	Output analog value with digital/analog converter of analog interface option

Relay Interface

REL nOUT "R", bitpattern	Switch one or several relays according to variable bit pattern
REL nOUT RelNumber$, state	Set relay RelNumber$ to switched state (n: interface number 1 to 4)

TTL-I/O-Interface

TTL a IN "port 1 [port 2]", V1	Reading-in via ports A to E
TTL a OUT "port 1 [port 2]", b	Output via ports A to E
TTL a IN "F", b	Reading-in via port F
TTL a OUT "F", b	Output via port F
TTL a IN "HSK", V1	Checking the handshake status
TTL a OUT "HSK", 0	Initialize input for handshake
TTL a OUT "HSK", 1	Initialize output for handshake
TTL a OUT "INT"	Interrupt with TTL I/O interface
TTL a IN "INT", V1	Detection of interrupt line
TTL a IN "port bit", V1	Reading-in of single bits via ports A to E
TTL a OUT "port bit", b	Output of single bits via ports A to E
TTL a OUT "2.5 V"	Setting the internal voltage source

Universal Control Interface

UCI a IN "port 1 [port 2]",V1	Poll port groups A to E
UCI a OUT "port 1 [port 2]",V1	Set port groups A to E
UCI a OUT "INT"	Interrupt with TTL I/O interface
UCI a IN "INT",V1	Determine line causing interrupt
UCI a IN "port bit",V1	Read in single bits
UCI a OUT "port bit",V1	Set single bits for port groups A to E
UCI a OUT "2.5V"	Set internal voltage source

IEC-bus instructions

Universal instructions

IEC ATN	ATN line active
IEC NATN	ATN line passive
IEC DCL	Device clear
IEC EOI	Output terminator with EOI
IEC NEOI	Output terminator without EOI
IEC IFC	Transmit IFC
IEC LLO	Local Lockout
IEC PPD	Parallel poll disable
IEC PPE k_1 k_2	Parallel poll enable
IEC PPL v%	Parallel poll
IEC PPU	Parallel poll unconfigure
IEC RLC	Release control
IEC RQS	Send service request
IEC REN	REN line active
IEC NREN	REN line passive
IEC SPD	Serial poll disable
IEC SPE	Serial poll enable
IEC TERM a	Define input terminator
IEC TIME a	Set time-out monitor
IEC T1	Set time T1
IEC UNL	Unaddress the devices as listeners
ON SRQ GOSUB m ON SRQ GOTO n	Jump on SRQ
ON SRQ RETURN	Inhibit jump upon SRQ

Addressed instructions

IEC ADR a	Assign address
IEC GET	Group execute trigger
IEC GTL	Go to local
IEC IN a_1,[;a_2],v$	Read in data
IEC $IN v$	Enter character string without address
IEC %IN v%	Enter character without address
IEC LAD a	Transmit listener address
IEC LIST ON	Switch on parallel output on IEC bus
IEC LIST OFF	Switch off parallel output on IEC bus
IEC MTA (IECUNT)	Transmit untalk
IEC OUTa_1[;a_2],s$[;]	Transmit character string
IEC $OUT s$	Output character string without address
IEC %OUT a%	Output individual character
IEC PCON b,k_1;k_2	Parallel poll configure
IEC PPC	Parallel poll configure
IEC SAD a	Transmit secondary address
IEC SDC	Selected device clear
IEC SPL b_1[;b_2]$_0$v%	Serial poll
IEC TAD a	Transmit talker address
IEC TCT	Take control
IEC UNL	Transmit unlisten
IEC WMLA	Wait for listener address
IEC WMTA	Wait for talker address
IEC WTC	Wait for takeover of control

Error messages

Error message:	Type of fault:
ERROR 1: "hardware not installed"	Interface addressed by the software is not fitted.
ERROR 2: "COM: DSR not active"	External V 24 device is not ready or not connected to the PCA.
ERROR 3: "COM: timeout"	Timeout for handshake or data reception on V 24 interface.
ERROR 4: "COM: overrun"	Input buffer of the V 24 interface contains more than 512 characters.
ERROR 5: "COM: parity"	Error in test bit in data read by V 24 interface.
ERROR 6: "COM: framing"	Stop bit has not been received with data input to V 24 interface.
ERROR 7: "device not open"	An attempt has been made to read from or write to an interface which has not been opened with OPEN.
ERROR 8: "device driver not installed"	Device driver of the operating system accessed by the incorrectly terminated instruction has not been loaded in CONFIG. SYS.
ERROR 9: "subroutine not loaded"	A subroutine number has been output with CALL # which has not yet been loaded.
ERROR 10: "IEC-bus timeout"	IEC-bus waiting time exceeded.
ERROR 11: "IEC-bus handshake error"	Error in IEC-bus handshake on NDAC and NRFD
ERROR 12: "not an IEC-bus talker/listener"	BASIC is not in the IEC-bus talker/listener status
ERROR 13: "not an IEC-bus controller"	BASIC is not in the IEC-bus controller status and must therefore not execute the instruction
ERROR 14: "I/O-control param. out of range"	String contains out-of-range values
ERROR 15: "I/O-control syntax"	String for setting the input/output interface is faulty.
ERROR 16: "device: general failure"	Interface for input/output signals a failure.
ERROR 17: "device: time out"	Input/output via the interface is not possible within the given period of time.
ERROR 18: "too deeply nested"	Maximum permissible number of subroutine levels has been exceeded.
ERROR 19: "FOR without TO"	FOR NEXT loop is not completely defined.
ERROR 20: "redimensioned array"	This data field has already been defined.
ERROR 21: "duplicate label"	Label has been used twice

ERROR 22: "incompatible version"	e. g. is not compatible with this BASIC version
ERROR 23: "GOSUB without RETURN"	Subroutine call without return instruction.
ERROR 24: "line number > 65534"	The highest line number 65534 has been exceeded.
ERROR 25: "undefined line or label"	Branch instruction to a non-existent program line.
ERROR 26: "FOR without matching NEXT"	The NEXT instruction is missing in the FOR NEXT loop.
ERROR 27: ") and (out of balance"	Incorrect combination of ( )
ERROR 28: "undefined command"	The entry does not correspond to an existing instruction.
ERROR 29: "out of memory"	Permissible memory area violated.
ERROR 30: "undefined variable"	Variable used has not been previously dimensioned.
ERROR 31: "numeric overflow"	Permissible number range of controller has been exceeded, e. g. when converting into an integer.
ERROR 32: "subscript out of range"	Field index outside the permissible range limits.
ERROR 33: "illegal math. operation"	Illegal mathematical operation.
ERROR 34: "RETURN without GOSUB"	Return from a subroutine not called by GOSUB.
ERROR 35: "syntax error"	Faulty instruction; brackets, letters or characters are wrong.
ERROR 36: "lines nested"	Illegal RENUMBER or CHAIN statement.
ERROR 37: "variable type mismatch"	Wrong type of variable used.
ERROR 38: "undefined operator"	Undefined variable within an instruction.
ERROR 39: "out of data"	An attempt has been made to read in more data than are present.
ERROR 40: "parameter too large"	Parameter outside the permissible range.
ERROR 41: "line too long"	The permissible number of 80 characters within a line has been exceeded.
ERROR 42: "illegal quantity"	Permissible number range of the operation has been exceeded.
ERROR 43: "can't continue"	Program cannot be continued with "CONT".
ERROR 44: "parameter out of range"	Permissible parameter range has been violated.
ERROR 45: "CHAIN line erase"	CHAIN line has been overwritten by the program to be loaded.
ERROR 46: "Pascal not loaded"	Pascal cannot be loaded.
ERROR 47: "undef'd function"	Function call (FN) without previous DEF FN.
ERROR 48: "no IF-struc., [ELSE], ENDIF match"	Structure is not complete.
ERROR 49: "file already open"	An attempt has been made to open a file which is already open.
ERROR 50: "file not open"	An attempt has been made to use an unopened file.
ERROR 51: "DOS close error"	Error when closing a file.
ERROR 52: "DOS open error"	Error when opening a file, e. g. because it does not exist on this subdirectory.
ERROR 53: "DOS write error"	Error when recording on floppy disk.
ERROR 54: "no valid file number"	Reference number of the file is not within the permissible range.
ERROR 55: "file not open for output"	File has not been opened as an output file.
ERROR 56: "file not open for input"	File has not been opened as an input file.
ERROR 57: "file name error"	Permissible length of the filename with 8 letters before the point and 3 letters after it has been exceeded.
ERROR 58: "DAC output overflow"	Refers to ANG interface.
ERROR 59: "ADC input overflow"	Refers to ANG interface.
ERROR 60: "WHILE without WEND"	Structure is not complete.
ERROR 61: "WEND without WHILE"	Structure is not complete.
ERROR 62: "REPEAT without UNTIL"	Structure is not complete.
ERROR 63: "UNTIL without REPEAT"	Structure is not complete.
ERROR 64: "DOS: write protected disk"	Floppy disk is write protected.
ERROR 65: "DOS: unknown unit"	*)
ERROR 66: "DOS: drive not ready"	No floppy disk inserted in the drive.
ERROR 67: "DOS: unknown command"	*)
ERROR 68: "DOS: data error"	*)

ERROR 69:
"DOS: bad request s. length" *)

ERROR 70: "DOS: seek error" *)

ERROR 71:
"DOS: unknown media type" Wrong floppy disk format.

ERROR 72:
"DOS: sector not found" Defective floppy disk or Winchester.

ERROR 73:
"DOS: printer out of paper" Printer out of paper, not switched on or not selected.

ERROR 74: "DOS: write fault" *)

ERROR 75: "DOS: read fault" *)

ERROR 76: "DOS: general failure" *)

ERROR 77:
"protected file loaded" The loaded program is not to be listed and cannot be changed

ERROR 78:
"too many GOTO/GOSUBs" Too much branching (memory overflow)

ERROR 79:
"too many open files" Opening of further files or interfaces not possible

ERROR 80:
"function not allowed here" Variable name must not begin with FN (function call!)

---

*) Error messages of the operating system.

ASCII/ISO- & IEC-CODE CHART

CONTROL						NUMBERS SYMBOLS				UPPER CASE				LOWER CASE			
0	NUL		16	DLE		32	SP	48	0	64	@	80	P	96	`	112	p
1	SOH	GTL	17	DC1		33	!	49	1	65	A	81	Q	97	a	113	q
2	STX		18	DC2		34	"	50	2	66	B	82	R	98	b	114	r
3	ETX		19	DC3		35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	SDC	20	DC4	DCL	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	PPC	21	NAK	PPU	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK		22	SYN		38	&	54	6	70	F	86	V	102	f	118	v
7	BEL		23	ETB		39	'	55	7	71	G	87	W	103	g	119	w
8	BS	GET	24	CAN	SPE	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	TCT	25	EM	SPD	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF		26	SUB		42	*	58	:	74	J	90	Z	106	j	122	z
11	VT		27	ESC		43	+	59	;	75	K	91	[	107	k	123	{
12	FF		28	FS		44	,	60	<	76	L	92	\	108	l	124	|
13	CR		29	GS		45	–	61	=	77	M	93	]	109	m	125	}
14	SO		30	RS		46	.	62	>	78	N	94	^	110	n	126	~
15	SI		31	US		47	/	63	? UNL	79	O	95	_	111	o	127	DEL
ADDRESSED COMMANDS			UNIVERSAL COMMANDS			LISTEN ADDRESSES				TALK ADDRESSES				SECONDARY ADDRESSES OR COMMANDS			

Key: